Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Sedej

Namizna lučka 'Ambilight'

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: viš. pred. dr. Robert Rozman

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

O razsvetljavi v bivalnih okoljih in njenem vplivu na človeka je vedno več novih spoznanj. Uporaba tehnologije LED na tem področju omogoča velike prihranke in več svobode pri njeni zasnovi. Naredite prototip namizne lučke, ki omogoča prijetno uporabniško izkušnjo pri upravljanju in hkrati upošteva najnovejša spoznanja v zvezi z vplivom ambientalne razsvetljave na človekovo počutje in cirkadiani ritem. Prototip izdelajte s pomočjo naprav, ki jih najdete na trgu. Dopolnite sistem z lastnimi rešitvami tam, kjer je to potrebno in smiselno. Sistem naj zagotovi uporabniku udobno, prijazno in cenovno dostopno upravljanje lučke. Ta naj omogoča tudi izdelavo avtomatiziranih opravil. Ob tem zagotovite še ustrezno povezljivost in čim preprostejše upravljanje sistema.

Zahvaljujem se svojemu mentorju viš. pred. dr. Robertu Rozmanu za strokovno vodenje pri izdelavi diplomske naloge.

Zahvaljujem se tudi Niki za pomoč in podporo pri izdelavi diplomskega dela.

Diplomsko delo posvečam staršema Mileni in Damijanu za vse spodbudne besede, podporo in pomoč v času študija.

Kazalo

Kazalo programskih kod

Slike

Seznam uporabljenih kratic

kratica	angleško	slovensko
HTML5	Hyper Text Markup Language 5	označevalni jezik
CSS	Cascading Style Sheets	kaskadne stilske predloge
SPA	Single-page application	enostranska aplikacija
MVC	Model-view-controller	model-pogled-krmilnik
NoSQL	Not only SQL	nerelacijska podatkovna baza
FIFO	First in first out	čakalna vrsta
RGB	red, green, blue	rdeča, zelena, modra
DOM	Document Object Model	dokumentno objektni model
IP	Internet Protocol	internetni protokol
GPIO	General Purpose Input/Output	splošno namenski vhod/izhod

Povzetek

Naslov: Namizna lučka 'Ambilight'

V sklopu diplomskega dela smo razvili prototip namizne luči, ki je s pomočjo brezžične mrežne kartice povezan v omrežje. Delovanje luči lahko nadziramo preko mobilne aplikacije na telefonu ali tablici z operacijskim sistemom Android. Luč lahko upravljamo tudi preko spletne aplikacije s kateregakoli računalnika v omrežju. Uporabnik lahko izbira med različnimi načini uporabe; med temi je najbolj pomemben način cirkadianega ritma, ki uporabniku zagotavlja zanj optimalno osvetlitev.

Zaledni del sistema, ki se odziva na uporabnikove akcije poganja računalnik Raspberry Pi, ki preko razvitega aparaturnega prototipa posledično krmili našo namizno luč.

Pri našem delu smo najprej raziskali vpliv svetlobe različnih barv na človekovo počutje; med njimi smo se osredotočili predvsem na oranžno in modro svetlobo. Zasnovali smo tudi vso potrebno programsko, strojno opremo skupaj z vezalno shemo ter v ohišju namizne lučke realizirali sistem v obliki samostojnega prototipa.

Ključne besede: Raspberry Pi, AngularJS, Framework7, Python, Flask, Android, cirkadiani ritem, razsvetljava.

Abstract

Title: 'Ambilight' Table Lamp

In this thesis, a prototype of a table lamp has been developed. The operation of the lamp can be controlled on a smartphone or a tablet via a mobile application for the Android operating system. The lamp can also be controlled via a web application from any computer connected to the network. Users can choose between different modes of operation; among them, the most important is the circadian rhythm mode that provides the user with the optimal well-being lighting.

The back-end part of the system, which responds to the actions of the user, is powered by a Raspberry Pi that in turn controls the table lamp via designed hardware prototype board.

In our diploma thesis, we first studied the effect of light of different colors on the human being. Among them, we focused on the orange and blue light. We have also designed all the necessary software, hardware and a wiring diagram and have created the standalone product prototype in the existing table lamp housing.

Keywords: Raspberry Pi, AngularJS, Framework7, Python, Flask, Android, circadian rhythm, lighting.

Poglavje 1

Uvod

Pred prihodom umetne razsvetljave je bilo sonce edini vir svetlobe. Takrat so ljudje večere preživljali v temi ali ob svečah. Danes so nam osvetljeni večeri samoumevni.

V času hitrega tempa življenja in veliko stresa so digitalne naprave v moderni dobi tehnologije postale del našega vsakdana, brez njih si preprosto ne moremo več predstavljati življenja, saj smo jim izpostavljeni na vsakem koraku. V šolah, službah ter v našem popoldanskem času ob branju e-knjig, sedenjem za računalnikom, gledanju televizije ali brskanju po telefonu. Zaradi tega smo čedalje več časa izpostavljeni umetni in modri svetlobi, ki jo proizvajajo digitalne naprave ter nekatere druge naprave in luči, kar pa lahko vpliva tudi na naše počutje. Zaradi uživanja ob luči in digitalnih napravah plačujemo svojo ceno, saj ta svetloba vpliva na telesno biološko uro – cirkadiani ritem, ki vpliva na spanje in zdravje človeka.

Številne študije dokazujejo, da izpostavljenost modri svetlobi, ki jo oddajajo digitalne naprave ter določene luči in naprave v popoldanskem in večernem času škodujejo zdravju [1], zato smo se odločili, da naredimo namizno lučko, ki je vodena s pomočjo spletne ali mobilne aplikacije preko zalednega sistema. Sistem je nameščen na računalniku Raspberry Pi. Aplikacija omogoča poljubno izbiro barv, samodejno prelivanje med barvami, dodajanje časovnih dogodkov, kjer izberemo poljubno barvo ter način cirkadianega

ritma. Ta deluje s pomočjo že prej nastavljenih dogodkov s specifičnimi barvami ob točno določenih urah, ki blagodejno vplivajo na cirkadiani ritem.

V nadaljevanju bomo v drugem poglavju predstavili vpliv svetlobe na človekovo počutje. V tretjem poglavju so opisane uporabljene tehnologije in programska orodja pri izdelavi uporabniškega vmesnika. Ker je uporabniški vmesnik povezan z zalednim sistemom, bomo v tem poglavju opisali tudi tehnologije in programska orodja zalednega sistema. V četrtem poglavju bomo podrobneje opisali tudi izvedeno strojno opremo sistema. Zasnovo našega sistema bomo predstavili v petem poglavju, ki je sestavljeno iz ideje, izbire platforme in arhitekture sistema. V šestem poglavju bomo predstavili razvoj sistema, kjer bomo opisali izbiro podatkovne baze, razvojno okolje, razvoj zalednega sistema, uporabniškega vmesnika in razvoj strojnega dela. Diplomsko delo bomo zaključili s sklepnimi ugotovitvami.

Poglavje 2

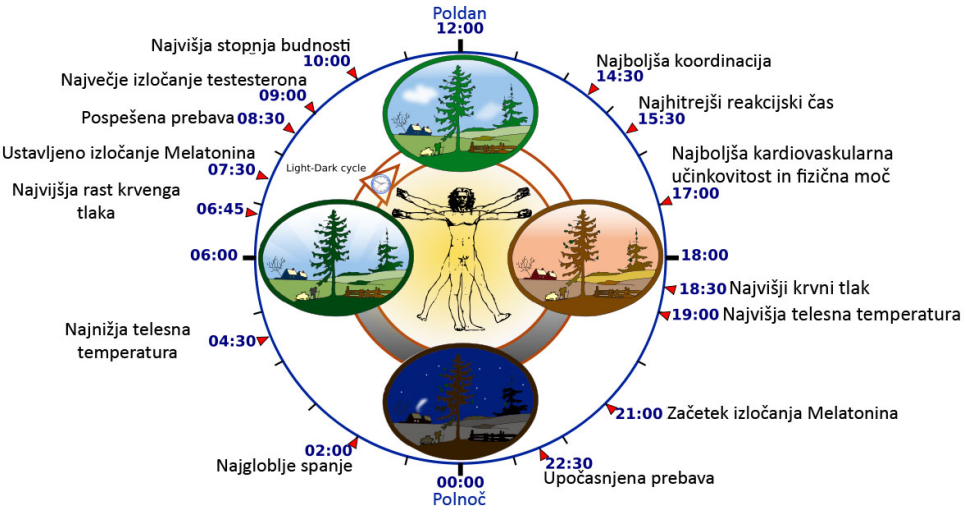
Vpliv svetlobe na človeka

V prvem delu bomo najprej predstavili cirkadiani ritem, da bomo v drugem delu lažje razložili, kakšne negativne in pozitivne vplive ima lahko svetloba na biološko uro človeka. V drugem delu bomo predstavili tudi, kakšen je vpliv modre svetlobe na človeka, ki jo oddajajo digitalne naprave in nekatere luči.

2.1 Cirkadiani ritem

Beseda cirkadiani izhaja iz dveh latinskih besed, in sicer besede circa, ki pomeni približno ter besede diem, ki pomeni dan. Če povežemo ti dve besedi, ugotovimo, da gre za proces, ki prikazuje spremembe tekom enega dneva. Ta 24 urni ritem je moč opaziti pri rastlinah, živalih in seveda ljudeh. Najbolj opazna stvar v ritmu pri ljudeh je proces spanja in budnosti, ki je samodejno nadzorovan s strani možganov. Če vse povzamemo, je cirkadiani ritem proces spreminjanja bioloških procesov v človekovem telesu tekom enega dneva. Na sliki 2.1 si lahko ogledamo pregled cirkadiane ure pri odraslem človeku, ki se zbudi zgodaj zjutraj, opoldne poje kosilo in odide spat ob 22. uri.

V kolikor se biološki procesi izvajajo v pravilnih časovnih okvirjih cirkadianega ritma govorimo o cirkadianem ravnotežju. Takrat smo zelo skoncentrirani, razigrani, polni energije ter zdravi. V kolikor pa pride do rušenja



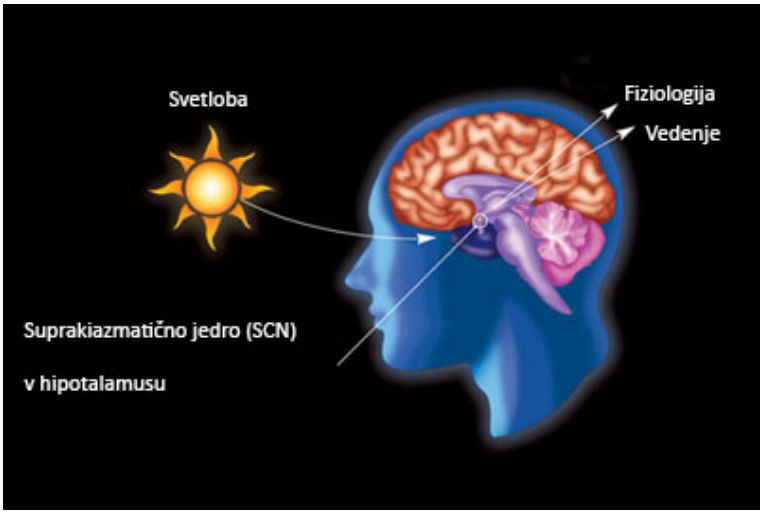
Slika 2.1: Prikaz cirkadianega ritma pri odraslem človeku [31].

cirkadianega ravnotežja se nam poslabša koncentracija, spanje, inteligenca itd. Zaradi tega lahko sklepamo, da ima rušenje cirkadianega ritma velik vpliv na naše zdravje [1].

2.2 Vpliv svetlobe na cirkadiani ritem

Cirkadiani ritem in od njega odvisni biološki procesi so prirojeni in imajo stalno periodo, ki je po navadi nekoliko daljša od 24 ur. Sprožanje bioloških procesov v pravilnih časovnih okvirjih sproži takt notranje ure s spodbujevalcem, ki se nahaja v suprakiazmatičnemu jedru v hipotalamusu. To je predel v možganih (slika 2.2), ki mu pravimo tudi centralna ura. Kot dokaz, da je suprakiazmatično jedro centralna ura, so hrčkom odstranili 75% suprakiazmatičnega jedra in s tem izničili izločanje hormona melatonina, ki je pri sesalcih glavni hormon spanja, vpliva pa tudi na imunski sistem. Dejavnost suprakiazmatičnega jedra sicer spodbuja osvetljenost očesne mrežnice, ki nato zavira hormon melatonin [1].

Velik vpliv na cirkadiani ritem ima tudi svetloba. Le-ta vpliva na raven kortizola (hormon budnosti) in melatonina. Slednji se začne izločati v



Slika 2.2: Pozicija suprakiazmatičnega jedra v možganih [31].

človeško telo ob 21. uri in se izloča do 7.30. Po 7.30 je priporočljivo, da smo izpostavljeni močni svetlobi, ki zelo hitro poveča raven kortizola in posledično poveča budnost. Svetloba pa lahko vpliva tudi na počutje ljudi. V kolikor je svetlobe ravno dovolj, ta deluje pozitivno in stimulirajoče. V primeru, da je svetlobe premalo, preveč ali pa ni pravilno razporejena, le-ta deluje negativno in utrujajoče. Poleg tega ustrezna osvetlitev poveča pozornost in aktivnost, medtem ko se pri neustrezni osvetlitvi pripravljenost za delo zmanjša [37].

Na to temo so naredili raziskavo tudi na belgijski fakulteti, kjer je sodelovalo 16 mladih ljudi med 18 – 30 letom starosti [38]. Sodelovalo je tudi 7 žensk. Raziskovalci so raziskovali, kakšen vpliv imata oranžna in modra barva na človekove kognitivne funkcije v možganih. Na vseh udeležencih je bilo opravljeno slikanje z magnetno resonanco. Med slikanjem so bili udeleženci osvetljeni z oranžno ali modro svetlobo, poleg tega pa so jih raziskovalci spraševali in s tem spremljali dogajanje v njihovih možganih. Raziskovalci so ugotovili, da so imeli udeleženci, ki so bili slikani pod oranžno svetlobo, večjo kognitivno možgansko aktivnost in povečano pozornost. To naj bi se zgodilo zaradi tega, ker ima oranžna barva (589 nm) večjo valovno dolžino, kot modra barva (461 nm). Izpostavljenost oranžni svetlobi skozi daljše ob-

dobje pozitivno vpliva na lažje jutranje zbujanje in boljšo zbranost čez dan.

Izpostavljenost modri ali katerikoli močni svetlobi pa ni priporočljiva po 21. uri, ko se v telo začne izločati melatonin. Kasneje je bolj priporočljivo, da svetlobo zamenjamo za nežno oranžno – rumenkasto svetlobo, saj izločanje melatonina zavira petkrat manj od modre svetlobe in trikrat manj od bele svetlobe. V primeru, da smo modri svetlobi izpostavljeni tudi po 21. uri, se v telo izloča malo melatonina, kar povzroči, da težje zaspimo. Posledici tega sta tudi nespečnost in rušenje cirkadianega ravnotežja. V kolikor dolgoročno prihaja do prekinitev cirkadianega ritma, to lahko privede do resnih bolezni srca in ožilja, v nekaterih primerih tudi srčnega infarkta, pojavijo se lahko še sladkorna bolezen in nekatere vrste raka. Te bolezni največkrat prizadenejo delavce, ki delajo v nočnih izmenah [2].

Na splošno je priporočljivo, da 90 minut pred spanjem nismo izpostavljeni modri svetlobi. V ameriški raziskavi [3], v kateri je sodelovalo 12 zdravih odraslih ljudi, so testirali, kakšna je kakovost spanja, jutranja budnost in vpliv na cirkadiani ritem po branju knjige v papirnati obliki in e-knjigi štiri ure pred spanjem. Testirance so razdelili v dve testni skupini. Obe skupini sta brali isto knjigo (ena skupina v papirnati obliki, druga v obliki e-knjige), brali sta jo pet dni zapored, v enakih časovnih intervalih. Ugotovili so, da so bili bralci knjige v papirnati obliki zaradi večjega izločanja melatonina po branju veliko bolj utrujeni kot bralci e-knjige. Bralci e-knjige so v povprečju porabili 10 minut več časa, da so zaspali, vendar so bili zjutraj bolj utrujeni kot bralci papirnatih knjig.

Rušenje cirkadianega ritma ne povzroča samo motenj v spanju, ampak vpliva tudi na naše zdravje. Če so poleg pomanjkanja spanca prisotna še telesna nedejavnost, neredna in neustrezna prehrana in stres, obstaja velika verjetnost, da bo prišlo do metabolnih motenj. V to skupino motenj spadajo: visoka raven holesterola, maščob in glukoze v krvi, visok krvni tlak, kopičenje maščobnega tkiva v predelu trebuha itd. Poleg tega tudi obstaja višja verjetnost srčnih kapi ter rakavih obolenj. Sploh pri ženskah, ki opravljajo nočno delo, je velika verjetnost obolenja raka na dojkah. To naj bi bilo povezano

s prekinjenim izločanjem hormona melatonina, ki se izloča ponoči. So pa znanstveniki dokazali, da je možnost obolenja za rakom pri slepih ženskah, ki ne zaznavajo svetlobe, manjše [1].

Poglavje 3

Tehnologije in orodja

To poglavje je sestavljeno iz opisov tehnologij in orodij, ki smo jih uporabili pri izdelavi uporabniškega vmesnika in zalednega sistema v sklopu našega diplomskega dela.

3.1 Uporabniški vmesnik

Pri razvijanju uporabniškega vmesnika smo uporabili več različnih tehnologij, zato bomo najpomembnejše opisali v nadaljevanju.

3.1.1 Označevalni jezik HTML5

Hyper Text Markup Language 5 (HTML5) je zadnja verzija tega označevalnega jezika, ki ga uporabljamo za predstavitev vsebine na svetovnem spletu, največkrat v obliki spletne strani. Te so z izidom zadnje verzije postale še zanimivejše, saj lahko sedaj na spletne strani dodajamo tudi multimedijsko vsebino, kot sta video in avdio, podpira pa tudi vektorsko grafiko. Povezovanje HTML5 dokumentov s stilskim jezikom CSS, pa nam omogoča, da lahko spletne vsebine prilagodimo tudi na različne mobilne naprave [4].

3.1.2 Kaskadne stilske predloge CSS

Cascading Style Sheets (CSS) je jezik za oblikovanje dokumentov, ki so napisani v označevalnem jeziku [5]. Po navadi se ga uporablja za oblikovanje uporabniških vmesnikov spletnih strani, z izidom verzije 3 pa tudi v mnogih mobilnih aplikacijah, ki so napisane v HTML-ju. Namen je, da s pomočjo selektorjev, razredov ter unikatnih identifikatorjev določamo stile elementom v označevalnem jeziku in s tem ločimo strukturo strani od njene predstavitve. S tem se je drastično zmanjšalo ponavljanje kode, kar je privedlo do manjših velikosti datotek, boljše preglednosti ter lažjega urejanja.

3.1.3 Programski jezik JavaScript

Dandanes je skoraj v vseh spletnih straneh poleg HTML-ja in CSS-ja prisoten tudi JavaScript, ki spletnim razvijalcem pomaga pri ustvarjanju interaktivnih spletnih strani. Razvilo ga je podjetje Netscape, danes pa je standardiziran po specifikaciji ECMAScript 6 [6]. Ker veliko spletnih brskalnikov še vedno ne podpira funkcionalnosti ECMAScript 6, se pri razvoju po navadi uporablja Babel, ki pretvori ECMAScript6 sintakso v ECMAScript 5 iz leta 2009. Popularnost JavaScripta je še narasla z nastankom knjižnice jQuery, ki jo bomo opisali v nadaljevanju. K popularnosti tega visokonivojskega jezika sta pripomogli tudi podjetji Google in Facebook z nastankom tehnologij, AngularJS in React, ki ju uporablja velik delež JavaScript razvijalcev.

JavaScript pa ni prisoten samo na spletu, ampak se uporablja tudi pri razvijanju iger, spletnih ter mobilnih aplikacijah, v zadnjem času pa je prisoten tudi v zalednih sistemih v jeziku Node.js [7].

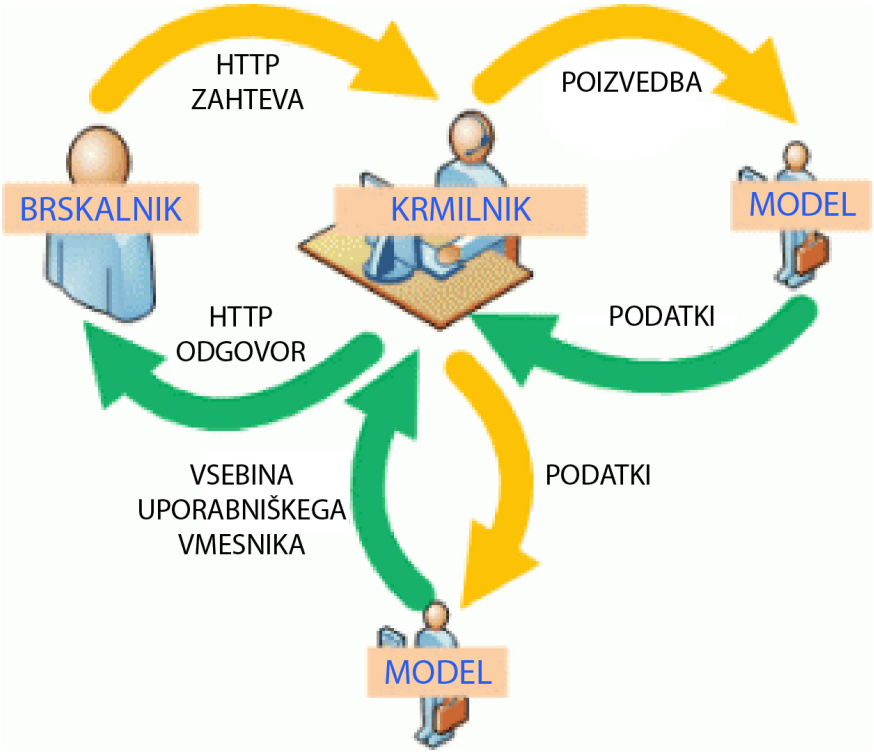
3.1.4 Knjižnica jQuery

JQuery je brezplačna odprtokodna JavaScript knjižnjica, ki je nastala leta 2006 [8]. Kljub temu je še vedno ena izmed najbolj priljubljenih JavaScript knjižnic, saj jo uporablja kar 65 % spletnih aplikacij. K temu je pripomogla enostavna sintaksa, ki s hitrim izbiranjem DOM elementov, kreiranjem

animacij, odzivi na dogodke in povezavo z zalednim sistemom razvijalcem omogočila lažje in hitrejše razvijanje dinamičnih spletnih strani in aplikacij.

3.1.5 Ogrodje AngularJS

AngularJS je odprtokodno JavaScript ogrodje, ki ga je razvilo podjetje Google. Namenjen je predvsem za razvijanje SPA aplikacij. To je spletna aplikacija ali spletna stran z eno samo stranjo, za katere je značilno, da se za razliko od ostalih aplikacij vsa koda (HTML, CSS, JavaScript) naloži samo enkrat. V odvisnosti od uporabniških akcij, pa se vsebina SPA aplikacije, ki jo po navadi dobimo iz zalednega sistema, s pomočjo usmerjevalnika spreminja dinamično [9]. Poleg tega AngularJS temelji na arhitekturi MVC (slika 3.1) [10].



Slika 3.1: Primer delovanja MVC arhitekture [32].

Arhitektura MVC je sestavljena iz modela, pogleda in krmilnika. Arhitektura MVC deluje tako, da krmilnik pridobi http zahtevo iz brskalnika. Na dano zahtevo krmilnik pošlje poizvedbo na model, kot rezultat pa vrne ustrezne podatke. Le-te krmilnik nato posreduje pogledu, ta pa obdelane podatke pošlje uporabniškemu vmesniku v obliki http odgovora.

AngularJS je sestavljen iz krmilnika, podatkovnih skladišč in direktiv. Že iz imena je razvidno, da Angularjev krmilnik predstavlja krmilnik v arhitekturi MVC. Pogled je uporabniški vmesnik z vgrajenimi direktivami, ki se prožijo ob uporabnikovih akcijah. Nato krmilnik pošlje poizvedbo v podatkovno skladišče, ki osveži podatke v modelu [10].

3.1.6 Ogrodje Framework7

V preteklih letih se je pojavilo veliko HTML ogrodij. Eno izmed njih je tudi Framework7, ki nam omogoča, da s pomočjo CSS-ja, HTML-ja in JavaScripta naredimo hibridno mobilno ali spletno aplikacijo, ki bo delovala na operacijskem sistemu IOS in Android. Ogrodje vsebujejo tudi privzete gradnike operacijskih sistemov, tako da lahko uporabimo komponente, kot so, meni, zaslonski meniji, drsniki itd., ki se prikažejo v privzeti obliki operacijskega sistema [11].

3.2 Zaledni sistem

Zaradi potreb po dinamičnem spreminjanju podatkov v uporabniškem vmesniku smo morali narediti tudi zaledni sistem. V nadaljevanju bodo opisane tehnologije, ki so bile uporabljane pri razvoju zalednega sistema.

3.2.1 Programski jezik Python

Python je eden izmed najbolj popularnih programskih jezikov, uporabljen je bil v vrsto uspešnih programih in aplikacijah, kot npr. Youtube in Instagram. Razvil ga je Guido van Rossum. Sintaksa Pythona je bila načrtovana

tako, da programerjem zaradi vgrajenih visokonivojskih podatkovnih struktur in elementov funkcijskega programiranja omogoča hitro programiranje ter lepo in berljivo kodo z manj vrsticami kot bi jih potrebovali v drugih programskih jezikih. Poleg tega Python omogoča tudi objektno orientirano programiranje [12].

Jedro filozofije jezika je povzeta po dokumentu The zen of Python (PEP 20) [13], ki vključuje aforizme, kot so:

- lepše je boljše kot grše,
- eksplicitno je boljše kot implicitno,
- enostavno je boljše kot kompleksno,
- kompleksno je boljše kot zakomplicirano,
- bralnost šteje.

Prednost Pythona je tudi, da je izjemno razširljiv jezik. Tako lahko na internetu poiščemo vrsto uporabnih knjižnic, ki jih lahko s pomočjo paketnega upravitelja Pip [14], enostavno naložimo ter uporabimo z nekaj enostavnimi ukazi.

Poleg vsega je Python prisoten tudi v spletnih tehnologijah, kjer se je sprva uporabljal samo pri zalednih sistemih. Z razvojem ogrodij, kot so Django, Pyramid in Flask, pa lahko sedaj Python umestimo tudi v HTML kodo.

3.2.2 Ogrodje Flask

Flask je mikro ogrodje za programski jezik Python in je namenjen spletnim razvijalcem. Leta 2010 ga je razvil Armin Ronacher, kot posledica prvo aprilske šale, ko je dejal, da bo nekoč naredil popularno spletno ogrodje. Za Flask govorimo, da je mikro ogrodje, ker ne zahteva posebnih orodij in knjižnic, saj ne vključuje funkcionalnosti za upravljanje s podatkovnimi bazami, obrazci

itd. Omogoča pa, da v ogrodje namestimo razširitve, ki jih v ogrodju potrebujemo. Dobra stvar razširitev je, da se posodabljajo hitreje, kot jedro samega ogrodja. Zaradi tega je Flask eno izmed najbolj popularnih Python ogrodij. Pri razvoju spletne aplikacije, sta ga uporabili tudi podjetji Pinterest in Linkedin [15].

V spodnjih alinejah so predstavljene nekatere dobre strani Flaska:

- vsebuje razvijalski strežnik in razhroščevalnik,
- ima integrirano podporo za teste,
- uporablja Jinja2 predloge,
- kompleksno je boljše kot zakomplicirano,
- podpira varne piškotke.

V programski kodi 3.1. si lahko ogledamo primer najbolj preproste aplikacije v Flasku, ki nam v spletnem brskalniku izpiše napis "Hello World".

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

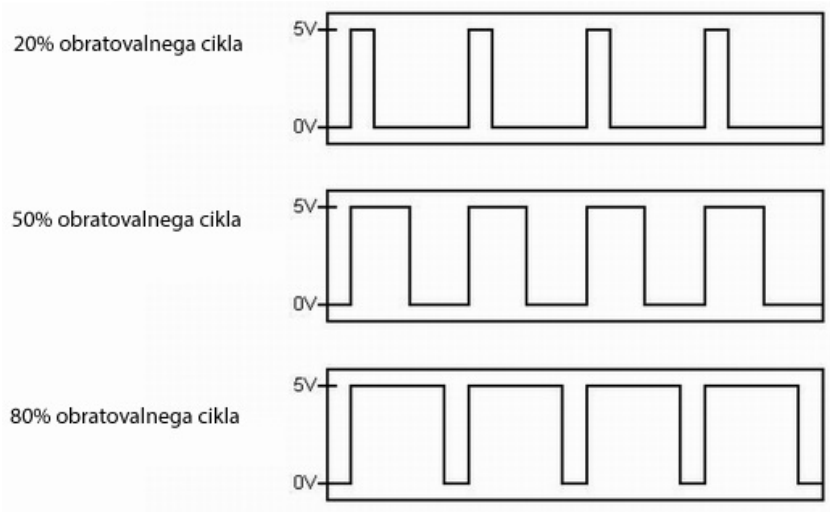
Programska koda 3.1: Primer enostavne Flask aplikacije.

3.2.3 Knjižnica APScheduler

APScheduler je knjižnica, ki je napisana v Pythonu in nam omogoča, da se del programske kode izvede z zamikom, samo enkrat ali pa periodično. Ker knjižnica omogoča, da opravila shranjujemo tudi v podatkovni bazi, lahko le-te dodajamo ali brišemo dinamično. Ena izmed dobrih lastnosti je, da časovnik deluje nemoteno tudi v primeru izpada internetne povezave [20].

3.2.4 Program Pi-blaster

Za vse, ki imajo na računalniku Raspberry Pi opravka s pulzno-širinskimi modulacijami, je Pi-blaster nepogrešljiv program. Ta na računalniku Raspberry Pi deluje kot program v ozadju, ki nadzira več pulzno-širinskih modulacij na GPIO vmesnikih.



Slika 3.2: Primer pulzno-širinske modulacije [33].

Pulzno-širinska modulacija [19] je način prikazovanja signala preko sekvence pulzov. To določata obratovalni cikel in frekvenca, ki je fiksna, hkrati pa določa v kolikšnem času, bo pulzno-širinska modulacija opravila en cikel (slika 3.2). Medtem ko obratovalni cikel določa odstotek časa enega cikla, ko je signal na logični enici.

$$f_{PWM} = \frac{1}{T_{PWM}}. \tag{3.1}$$

Pi-blaster deluje tako, da v mapi "/dev/pi-blaster"ustvari posebno datoteko, ki deluje po metodi FIFO, v njo pa lahko zapisujejo vse aplikacije, ki delujejo na računalniku Raspberry Pi.

3.2.5 Podatkovna baza MongoDB

MongoDB, ki sodi v družino NoSQL podatkovnih baz so razvili v podjetju MongoDB Inc. ter izdali v kombinaciji licenc Apache ter GNU Affero General Public License. Za razliko od tradicionalnih relacijskih podatkovnih baz, je MongoDB dokumentno orientirana podatkovna baza, ki v dinamičnih shemah hrani dokumente v formatu BSON. S pomočjo tega formata je povezovanje med določenimi tipi aplikacij veliko lažje in hitrejše. Prednost NoSQL podatkovne baze MongoDB je tudi veliko lažje dodajanje novih atributov, v primerjavi s tradicionalnimi relacijskimi bazami [18].

3.2.6 Operacijski sistem Android

Android je od leta 2013 najpopularnejši mobilni operacijski sistem. Za takšno popularnost je najbolj zaslužno podjetje Google, ki je leta 2005 kupilo hitro rastoče podjetje Android Inc. Sprva je bil namenjen le za pametne telefone in tablice z ekranom na dotik, v zadnjem času pa ga je moč opaziti tudi na drugih elektronskih napravah. Poleg vsega je Android odprtokoden operacijski sistem, ki je zgrajen na Linuxovem jedru. Zaradi brezplačnega razvijalskega orodja Android Studio je razvijanje Android aplikacij, ki poteka v visokonivojskem jeziku Java, brezplačno. Aplikacije pa je moč objaviti na aplikaciji v oblaku, ki se imenuje Google Play. Občutno prednost tu imajo uporabniki, saj so programi na tej aplikaciji večinoma brezplačni [17].

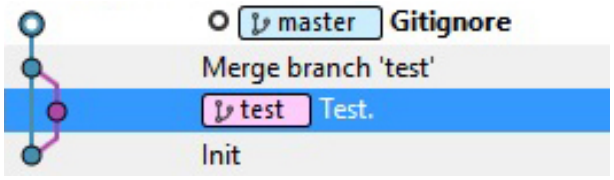
3.2.7 Programska oprema Vagrant

Vagrant je programska oprema, ki je namenjena ustvarjanju in konfiguriranju virtualnega razvojnega okolja. Lahko si ga predstavljamo kot višjo plast katerega izmed virtualnih programskih orodij ali programske opreme za upravljanje konfiguracije. Sprva se je Vagrant lahko uporabljal le v povezavi z VirtualBoxom. Z verzijo 1.1, pa ga lahko povežemo tudi z ostalimi virtualnimi programskimi orodji, ena izmed njih sta VMware in KVM. Vagrant je bil napisan v programskem jeziku Ruby, vendar ga je mogoče uporabiti

tudi v projektih, ki vključuje programske jezike Python, Java, JavaScript itd. [16]. Pred kreiranjem virtualnega razvojnega okolja, moramo najprej kreirati konfiguracijsko datoteko, v katero lahko vključimo ukaze in skripte za namestitev potrebnih programov in tehnologij, ki se nato poženejo ob ukazu `vagrant up`. Tako si lahko le z enim samim ukazom ustvarimo virtualno razvojno okolje, ki ima že vse nameščene tehnologije, ki jih potrebujemo pri razvoju našega sistema.

3.2.8 Sistem za upravljanje z izvorno kodo Git

Kolikokrat se nam je že zgodilo, da smo med razvojem ali v produkcijskem okolju želeli imeti del programske kode, ali pa s sodelavcem sodelovati in razvijati isti del programske kode. Rešitev za to je sistem Git, ki skrbi za kontrolo različic. Z njegovo pomočjo lahko prosto preklapljamo med shranjenimi različicami. S pomočjo tega se lahko pohitri tudi razvoj programske kode v podjetju, saj nam v vsakem trenutku omogoča, da lahko spremembo programske kode naložimo ali prenesemo iz skupnega Git repozitorija. Ena izmed prednosti je tudi, da lahko večje število razvijalcev razvija isti del programske kode. To pa storimo tako, da naredimo svojo vejo iz osnovne veje master in nato na to vejo shranimo svoje spremembe. Ko končamo z razvijanjem, nato našo vejo združimo nazaj v osnovno vejo. V tem koraku lahko pride do težave, če je medtem isti del programske kode spreminjal tudi nekdo drug, saj se lahko prepletata dve različni verziji. Zato moramo v tem koraku izbrati verzijo, ki jo bomo združili v osnovno vejo.



Slika 3.3: GIT - primer združevanja iz veje test v vejo master.

Na sliki 3.3 je predstavljeno delovanje sistema za kontrolo različic Git. V

našem primeru bomo iz osnovne veje master naredili vejo test, ki jo bomo po shranjeni spremembi zopet združili z osnovno vejo.

3.2.9 Orodje Virtualenv

Virtualenv [28] je orodje, ki se uporablja za kreiranje izoliranih Python okolij, ki so v današnjem programerskem svetu nujna. Predstavljamo si, da imamo na našem razvojnem okolju dve aplikaciji, ki uporabljata isto knjižnico, vendar sta različni verziji. Ko želimo verzijo knjižnice posodobiti lahko, kaj kmalu naletimo na težavo saj, je velika verjetnost, da je kakšna izmed funkcionalnosti, ki je prisotna v verziji 1 v naslednji verziji izbrisana. To pa privede do nedelovanja prve aplikacije. Zato uporabljamo orodje Virtualenv, ki nam omogoča, da knjižnice namestimo v izolirano Python okolje. Tako bi imeli za vsako aplikacijo svojo izolirano okolje, s tem pa rešimo težavo z odvisnostmi, verzijami in pravicami.

3.3 Programski orodji

Pri izdelovanju diplomskega dela smo izbrali dve programski orodji, in sicer Sublime Text [21] ter Android Studio [22]. Prvega smo uporabili pri razvoju zalednega sistema in uporabniškega vmesnika, saj podpira sintakse skoraj vseh programskih jezikov. Poleg tega lahko vanj naložimo tudi veliko drugih vtičnikov, ki nam pomagajo pri razvoju programske kode. Za vključitev uporabniškega vmesnika v mobilno aplikacijo, smo uporabili Android Studio, ki je privzeto programsko orodje za razvoj mobilnih aplikacij v Androidu.

Poglavje 4

Strojna oprema

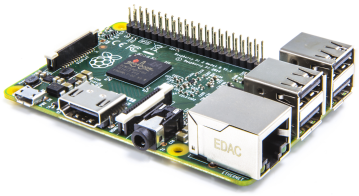
V četrtem poglavju bomo opisali strojno opremo, ki smo jo uporabili pri izdelavi namizne lučke, in sicer: računalnik Raspberry Pi, brezžično mrežno kartico, tranzistor MOSFET in svetlobni trak LED. Podrobnejši opis razvoja aparaturnega dela sistema je opisan v podpoglavju 6.5.

4.1 Računalnik Raspberry Pi

Britanska dobrodelna ustanova Raspberry Pi Foundation je za spodbujanje osnovnega računalniškega znanja v šolah leta 2012 razvila računalnik, ki je sestavljen iz enega samega čipa. Imenuje se Raspberry Pi (slika 4.1). Zaradi njegove velikosti in cene, zadnji model (Raspberry Pi 3), ki je izšel februarja 2016, stane le 35 dolarjev, je najbolje prodajan računalnik v Veliki Britaniji. Čeprav gre za računalnik v velikosti bančne kartice, lahko izbiramo med številni operacijskimi sistemi, od Linuxa pa do zadnje verzija Windowsa (Windows 10) [23].

Za potrebe našega diplomskega dela smo uporabili starejšo različico tega računalnika, in sicer Raspberry Pi 2 B. Le-tega smo uporabili, ker smo ga imeli na zalogi, poleg tega pa zadošča našim potrebam. Odločili smo se, da bomo na računalnik Raspberry Pi naložili operacijski sistem Raspbian, ki temelji na distribuciji Debian. Zaradi številnih priključkov lahko Raspberry

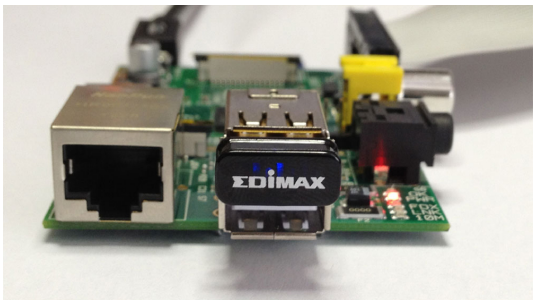
Pi povežemo tudi z zunanjem svetom, preko UTP kabla ali USB mrežne kartice in ga s tem spremenimo v spletni strežnik.



Slika 4.1: Raspberry Pi B 2 [34].

4.2 Brezžična mrežna kartica

V našem diplomskem delu računalnik Raspberry Pi uporabljamo kot spletni strežnik, zato moramo imeti dostop do lokalne internetne povezave. To lahko dosežemo s pomočjo USB mrežne kartice. V našem primeru smo se odločili za nano USB mrežno kartico Edimax EW-7811Un, ki je idealna za povezovanje računalnika Raspberry Pi z zunanjim svetom. Ta visoko zmogljiv USB adapter podpira višje hitrosti prenosa do 150 Mb/s po 802.11n [24] standardu in je kar trikrat hitrejši od standardne 11g povezave. Poleg tega je adapter varčen, uporabljamo ga lahko v zaščitenih omrežjih in je enostaven za uporabo [25].

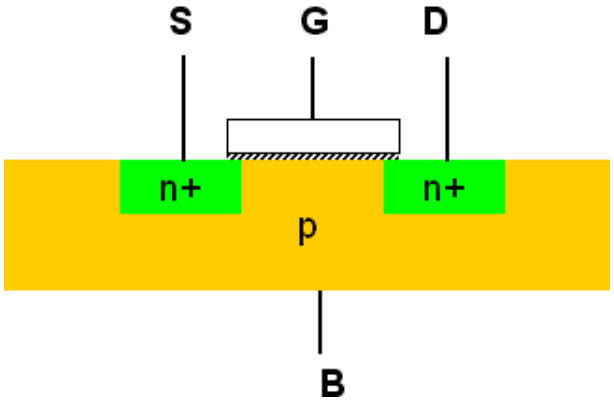


Slika 4.2: Nano USB mrežna kartica Edimax EW-7811Un [35].

4.3 Tranzistor MOSFET

Tranzistor je polprevodniški elektronski element, ki je sestavljen iz treh priključkov in je eden od ključnih gradnikov sodobne elektronike. Uporabljamo ga za ojačevanje, preklapljanje, uravnavanje napetosti itd. Za potrebe diplomskega dela smo uporabili tranzistorje MOFSET. Njegovo zgradbo si lahko ogledamo na sliki 4.5. Princip delovanja tega tranzistorja je drugačen, saj z ustvarjanjem električnega polja spreminjamo prevodnost njegovega glavnega kanala. Značilnost za tranzistorje MOSFET je tudi, da vsebujejo tanko plast kovinskega oksida [26]. Tranzistor MOSFET sestavljajo naslednji elementi:

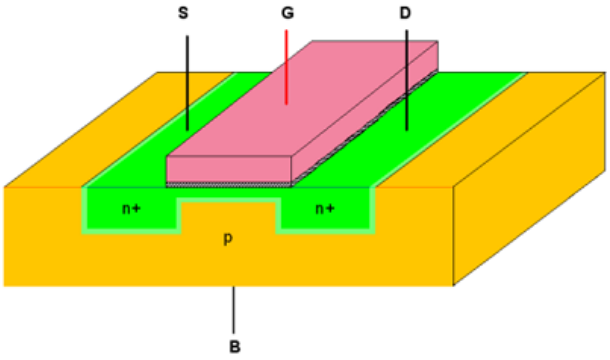
- S - izvor,
- D - ponor,
- G - vrata,
- B - substrat, podlaga,
- n+, p - silicij z različnimi vrstami primesi.



Slika 4.3: Zgradba tranzistorja MOSFET - prerez [36].

Pri tranzistorju MOSFET so vrata in izvor priključena na substrat. Na zgornji sliki električni tok med vrati in izvorom ni možen, saj substrat sam

ne prevaja. Vrata so od izvora izolirana s tanko plastjo kovinskega oksida. Vendar, če na vrata priključimo električno napetost se pod izolacijo ustvari prevodni kanal, ki omogoča električni tok med izvorom in ponorom. Tranzistor se v tem primeru uporablja kot stikalo, ki ga krmilijo vrata. Takšen primer si lahko ogledamo na sliki 4.4 [27].



Slika 4.4: MOSFET tranzistor s prevodnim kanalom [36].

4.4 Svetlobni trak LED

Svetlobni trak LED je ključen strojni del pri našem diplomskem delu, saj skrbi za svetilnost namizne lučke. Ker smo želeli, da bi namizna lučka prikazovala več barv smo izbrali RGB LED trak, ki je sestavljen iz 150 RGB diod. Posamezna dioda ima v notranjosti še tri majhne diode. Vsaka od njih oddaja različno barvo: rdečo, zeleno in modro. S spreminjanjem jakosti posameznega dela LED diode, lahko dobimo več kot 16 milijonov barvnih odtenkov svetlobe. Življenjska doba traku znaša 50.000 ur.

Poglavje 5

Zasnova sistema

Peto poglavje je namenjeno zasnovi sistema naše diplomske naloge. V prvem podpoglavju bomo podrobno opisali idejo naše diplomske naloge. V drugem podpoglavju bomo opisali, zakaj smo izbrali računalnik Raspberry Pi. V zadnjem podpoglavju bomo podrobno predstavili arhitekturo našega sistema.

5.1 Ideja

Odločili smo se, da bomo naredili namizno lučko, ki jo bo lahko uporabnik upravljal preko spletne in mobilne aplikacije. Uporabnik bo lahko poleg prižiganja in ugašanje lučke izbiral še med različnimi barvami. Za boljši ambient lahko izberemo program s prelivanjem barv. Spletna in mobilna aplikacija omogočata tudi beleženje dogodkov, ki jim določimo čas, kdaj bo svetilka svetila z izbrano barvo. Lahko pa si izberemo tudi način cirkadianega ritma, ki že vsebuje pred nastavljene dogodke z izbranimi barvami, ki blagodejno vplivajo na počutje človeka.

Ker se melatonin izloča do 7.30 zjutraj, v času med 6.45 - 7.30 lučka sveti v oranžni barvi ter simulira sončni vzhod. S tem želimo, da bi zelo hitro povečali raven kortizola in posledično povečali budnost. Po 7.30 se barva lučke spremeni v svetlo oranžno barvo, kar naj bi izboljšalo kognitivno možgansko aktivnost ter izboljšalo razpoloženje, počutje in pozornost. Od-

tenek svetlo oranžne barve se med enajsto in štirinajsto uro spremeni v toplo belo barvo. Lučka v tem odtenku sveti do 19. ure, ko se barva spremeni v svetlo oranžno. Ob 21. uri začne lučka svetiti v oranžni barvi. To barvo smo izbrali zato, ker zavira izločanje melatonina petkrat manj od modre svetlobe in trikrat manj od bele svetlobe. Ker je melatonin glavni hormon spanja pri sesalcih, postajamo ob izpostavljenosti oranžni svetlobi bolj zaspani, kot če bi bili izpostavljeni modri ali beli svetlobi. Zaradi tega bomo tudi lažje zaspali.

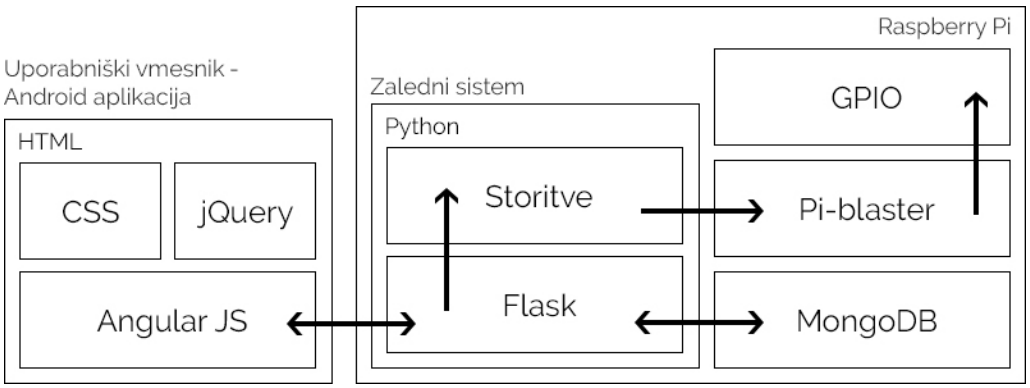
5.2 Izbira platforme

Za potrebe diplomskega dela smo si želeli izbrati platformo, ki bi znala upravljati z zunanjimi komponentami ter omogočala hiter in lahek razvoj. Odločili smo se za računalnik Raspberry Pi z operacijskim sistemom Raspbian. Ta kot privzeti programski jezik uporablja Python. Zaradi njegove enostavne sintakse je izjemno lahko napisati programsko kodo, ki preko GPIO vmesnika omogoča enostavno komunikacijo z zunanjimi komponentami. Poleg tega lahko z uporabo brezžične mrežne kartice računalnik Raspberry Pi povežemo z internetom in ga tako spremenimo v spletni strežnik. Raspberry Pi je zato idealna platforma za naš sistem, saj ga lahko uporabimo kot spletni strežnik in za komunikacijo z zunanjimi komponentami, poleg vsega pa je še cenovno ugoden.

5.3 Arhitektura sistema

Jedro našega sistema je računalnik Raspberry Pi. Ta ima v našem sistemu dve nalogi, in sicer, da deluje kot spletni strežnik ter krmili LED trak in tako skrbi za svetilnost naše namizne lučke. Na spletnem strežniku deluje zaledni sistem ter dokumentno orientirana podatkovna baza MongoDB. Zaledni sistem, ki je narejen s pomočjo Python ogrodja Flask, deluje v povezavi s podatkovno bazo, v kateri so shranjeni dogodki. Obdelane podatke, v našem

primeru so to dogodki, zaledni sistem posreduje uporabniškemu vmesniku, ki je narejen s pomočjo spletnih tehnologij, kot so: HTML, CSS, jQuery in AngularJS. Z uporabo spletnega pogleda je uporabniški vmesnik integriran v Android aplikacijo.



Slika 5.1: Arhitektura sistema.

Vsebina, ki jo vidimo na uporabniškem vmesniku je prikazana s pomočjo označevalnega jezika HTML, medtem ko je le-ta oblikovana s pomočjo stilskega jezika CSS. Uporabniški sistem z uporabo funkcij iz ogrodja AngularJS komunicira z zalednim sistemom, ki se odziva na uporabniške akcije. Če je uporabnik dodal, uredil ali izbrisal dogodek, bo zaledni sistem z uporabo ustreznih funkcij dodal, uredil ali izbrisal dogodek v podatkovni bazi MongoDB. Če pa je uporabnik spremenil barvo ali način delovanja na uporabniškem vmesniku, bo zaledni sistem sprožil storitev, ki s pomočjo knjižnice Pi-blaster spremeni barvo na LED traku. Barve lahko na uporabniškem vmesniku spreminjamo na gradniku, ki je narejen s pomočjo knjižnice jQuery. Arhitekturo celotnega sistema si lahko ogledamo na sliki 5.1.

Poglavje 6

Razvoj sistema

V najobširnejšem poglavju našega diplomskega dela bomo predstavili razvoj našega sistema. Podrobno bomo opisali izbiro podatkovne baze, razvojno okolje, razvoj zalednega sistema in uporabniškega vmesnika ter strojno opremo, ki smo jo zasnovali v našem diplomskem delu.

6.1 Izbira podatkovne baze

V sklopu našega diplomskega dela smo izbirali med relacijskimi in NoSQL podatkovnimi bazami. Po analizi podatkov smo se odločili za nerelacijsko podatkovno bazo MongoDB, saj je analiza pokazala, da v naši podatkovni bazi ne potrebujemo relacij. Zadostuje nam že preprosta zbirka dogodkov. Vsak dogodek je predstavljen kot dokument v BSON obliki, ki vsebuje ime, začetni ter končni čas, čas prehoda in barvo. Velika prednost MongoDB podatkovne baze je hitra implementacija v ogrodje Flask, saj jo je s pomočjo knjižnice Pymongo moč implementirati že z nekaj vrsticami programske kode. Za razliko od relacijskih baz, ki za začetek shranjevanja v podatkovno bazo ne potrebujemo kreirati tabel, saj podatkovna baza MongoDB zbirke kreira sama.

6.2 Razvojno okolje

V našem diplomskem delu smo uporabili dve okolji, razvojnega in produkcijskega. Za ločeni razvojni okolji smo se odločili predvsem zaradi lažjega ter hitrejšega razvijanja našega zalednega sistema. Ker produkcijsko okolje deluje na operacijskem sistemu Raspbian, ki temelji na distribuciji Debian, smo stremeli k temu, da naše razvojno okolje v čim večji meri približamo produkcijskemu. Zato smo sklenili, da s pomočjo programske opreme Vagrant ustvarimo virtualno razvojno okolje, ki bo temeljilo na distribuciji Debian.

Omenjeno distribucijo smo najprej poiskali na uradni spletni strani programske opreme Vagrant, nato pa jo vključili v konfiguracijsko datoteko, katere primer si lahko ogledamo v programski kodi 6.1.

```
Vagrant.configure(2) do |config|
  config.vm.box = "debian/jessie64"

  config.vm.network "forwarded_port", guest: 5000, host: 5000
  config.vm.network "forwarded_port", guest: 27017, host: 27017
  config.vm.network "private_network", ip: "192.168.33.10"
  config.vm.synced_folder "../diploma-data", "/home/vagrant/
    diploma"

  config.vm.provision "shell", inline: <<-SHELL
    sudo apt-get install build-essential -y
    sudo apt-get install python-setuptools -y
    sudo apt-get install ipython -y
    sudo easy_install pip
    sudo pip install virtualenv
    sudo apt-get install python-dev
    sudo apt-get install libevent-dev
    sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80
   --recv 7F0CEB10
    echo "deb http://repo.mongodb.org/apt/ubuntu "$(lsb_release
   -sc)"/mongodb-org/3.0 multiverse" | sudo tee /etc/apt/sources
    .list.d/mongodb.list
    sudo apt-get update
    sudo apt-get install mongodb-org -y
```

```
    sudo apt-get install git -y
    sudo apt-get update
    sudo apt-get upgrade -y
  SHELL
end
```

Programska koda 6.1: Konfiguracijska datoteka programske opreme Vagrant.

Programska koda 6.1 poleg distribucije vsebuje še IP, preko katerega lahko dostopamo do virtualnega razvojnega okolja. Poleg tega imamo omogočena tudi dvojna vrata, in sicer 5000 in 27017. Prva uporablja ogrodje Flask, druga pa podatkovna baza MongoDB. Konfiguracija določa tudi simbolično povezavo med mapama `diploma` in `diploma-data`. Slednja se nahaja na našem računalniku, njene datoteke pa so s pomočjo simbolične povezave preslikane v mapo `diploma`, ki se nahaja na virtualnem razvojnem okolju. Znotraj razvojnega okolja je mapa, dostopna na poti `/home/vagrant/diploma`. V nadaljevanju konfiguracije so vključeni tudi ukazi. Ti se izvedejo ob prvem klicu ukaza `vagrant up` potem, ko se prenese distribucija. V nadaljevanju se ta ukaz uporablja za vklop virtualnega razvojnega okolja, do katerega lahko dostopamo s pomočjo ukaza `vagrant ssh` (slika 6.1).

```
Uporabnik@Lenovo-PC MINGW64 ~/Desktop/test/diploma
$ vagrant ssh
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-105-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Tue Jan  3 16:57:51 UTC 2017

  System load:  0.0               Processes:           81
  Usage of /:   4.6% of 39.34GB   Users logged in:     1
  Memory usage: 32%               IP address for eth0: 10.0.2.15
  Swap usage:   0%                IP address for eth1: 192.168.33.10

vagrant@vagrant-ubuntu-trusty-64:~$ ls
diploma
vagrant@vagrant-ubuntu-trusty-64:~$ cd diploma/
vagrant@vagrant-ubuntu-trusty-64:~/diploma$ ls
ambilight     nginx.conf            start_server.sh  uwsgi_dev.ini
ambilight.py  requirements.txt      static           uwsgi.ini
env           start_server_backup.sh  templates      uwsgi.txt
vagrant@vagrant-ubuntu-trusty-64:~/diploma$
```

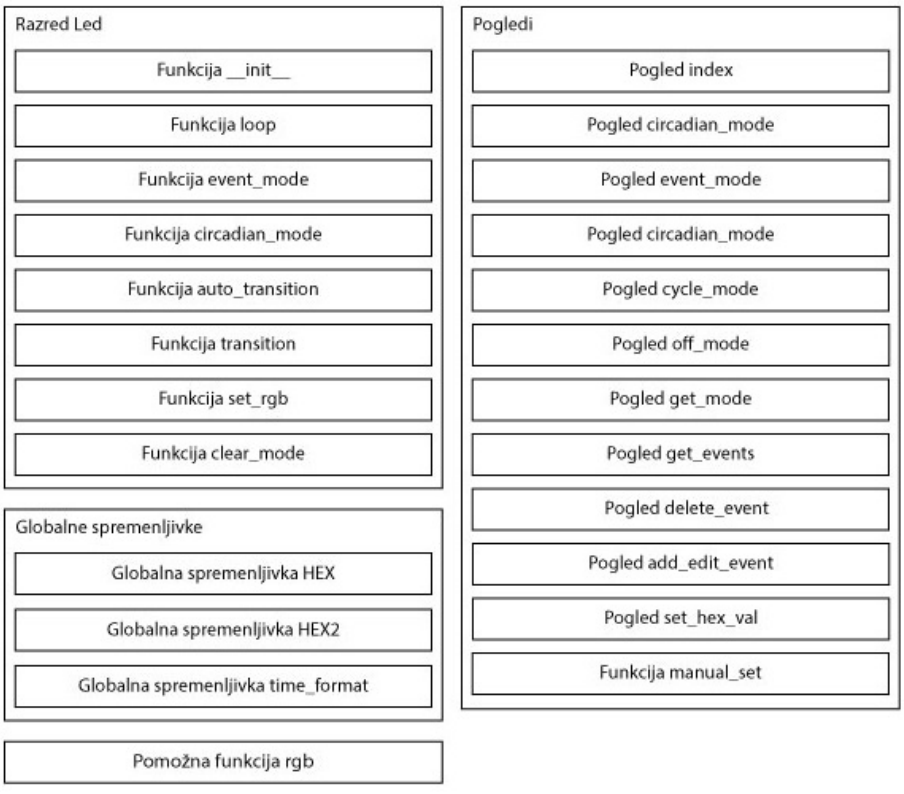
Slika 6.1: Povezava na virtualno razvojno okolje.

Po uspešno kreiranem virtualnem razvojnem okolju, moramo ustvariti še

izolirano okolje programskega jezika Python. Najprej z uporabo ukaznega poziva izvedemo ukaz `vagrant ssh`, s pomočjo katerega se povežemo na virtualno razvojno okolje, na katerem z ukazom `virtualenv env` ustvarimo izolirano okolje z imenom `env`. Tega moramo še aktivirati, pri čemer nam pomaga ukaz `source env/bin/activate`. Po izvedenem ukazu je naše okolje pripravljeno za uporabo.

6.3 Razvoj zalednega sistema

Kot je razvidno že iz naslova poglavja, bomo v tem poglavju opisali razvoj zalednega sistema. Na začetku poglavja bomo razložili, zakaj smo se odločili za uporabo Python ogrodja Flask.



Slika 6.2: Shema zalednega sistema.

V nadaljevanju pa bomo po korakih opisovali razvoj zalednega sistema. Shemo le-tega si lahko ogledamo na sliki 6.2.

6.3.1 Izbira programskega jezika in ogrodja

Za razvoj našega zalednega sistema smo želeli uporabiti moderen programski jezik s široko množico uporabnikov ter preprosto sintakso. Zaradi teh razlogov smo izbrali programski jezik Python, ki ga krasijo preprosta sintaksa, berljivost ter razširljivost. Python je zato eden izmed najbolj priljubljenih spletnih programskih jezikov. Za lažje in preprostejše razvijanje spletnih aplikacij so se pojavila tudi najrazličnejša ogrodja.

Izbirali smo med najpopularnejšimi, med katere sodijo Django, Flask in Pyramid. Pyramid in predvsem Django sta ogrodji, primernejši za večje projekte, ki vsebujejo avtentikacijo, avtorizacijo, administratorsko ploščo itd., saj v ta namen vsebujejo večje število modulov. Flask za razliko od omenjenih ogrodij vsebuje manjše število modulov, a ravno pravšnje za hiter in preprost razvoj spletnih aplikacij, zaradi tega je Flask mikro ogrodje. Uporabimo ga lahko z vsemi podatkovnimi bazami. Na internetu najdemo veliko število modulov, ki jih nato vključimo v Flask, zato smo se odločili, da bomo to ogrodje uporabili pri razvoju našega zalednega sistema.

6.3.2 Namestitev in uvoz knjižnic

Razvoj zalednega sistema smo začeli z namestitvijo knjižnic programskega jezika Python, med katerimi je bilo tudi ogrodje Flask. Le-tega smo namestili s pomočjo paketnega upravitelja Pip z uporabo ukaza `pip install Flask`. Ker bomo izolirano okolje ustvarili tudi na produkcijskem okolju je zaželeno, da si knjižnice in njihove verzije shranimo v datoteko, s pomočjo katere jih bomo kasneje namestili. Z ukazom `pip freeze` si nameščene knjižnice v izoliranem okolju izpišemo, nato pa le-te z ukazom `pip freeze > requirements.txt` shranimo v tekstovno datoteko z imenom requirements. V programski kodi 6.2 si lahko ogledamo vsebino naše datoteke.

```
APScheduler==3.2.0
Flask==0.10.1
Flask-PyMongo==0.4.1
Flask-Triangle==0.5.4
funcsigs==1.0.2
functools32==3.2.3.post2
futures==3.0.5
itsdangerous==0.24
Jinja2==2.8
jsonschema==2.5.1
MarkupSafe==0.23
pymongo==3.2.2
pytz==2016.4
six==1.10.0
tzlocal==1.2.2
Werkzeug==0.11.9
```

Programska koda 6.2: Vsebina datoteke requirements.txt.

Z namestitvijo knjižnic imamo vse pripravljeno za razvijanje zalednega sistema. Naslednji korak pri razvoju je kreiranje Python datoteke, v kateri bo delovalo ogrodje Flask. Ta se v našem primeru imenuje `ambilight.py`. V datoteki najprej vključimo knjižnice ter module, ki jih bomo uporabili tekom razvoja (programska koda 6.3).

```
import operator
import os
import pytz
import sys
import time
import Queue

from apscheduler.schedulers.background import
    BackgroundScheduler
from datetime import datetime, timedelta
from flask import Flask, jsonify, render_template, request
from flask.ext.triangle import Triangle
from sys import exit
from threading import Thread
```

```
from pymongo import MongoClient
from bson.json_util import dumps
from bson.objectid import ObjectId
```

Programska koda 6.3: Vključene knjižnice in moduli v datoteki ambilight.py.

Poleg Flaska uporabljamo še veliko drugih knjižnic. Med najpomembnejšimi so še Pymongo, Apscheduler, Datetime, Queue in Threading. Knjižnico Pymongo uporabljamo za povezovanje ogrodja Flask s podatkovno bazo MongoDB v kateri shranjujemo dogodke. Apscheduler nato poskrbi, da dogodek ob pravem času postavi v vrsto, ki teče na niti. Vrsta in nit sta implementirana s pomočjo dveh knjižnic: Queue in Threading. Časovne dogodke pa lahko obdelujemo s pomočjo knjižnice Datetime.

6.3.3 Inicializacija Flask ogrodja in podatkovne baze

Sedaj, ko imamo vključene vse knjižnice lahko nadaljujemo z razvojem našega programa. V programski kodi 6.4 bomo najprej definirali Flask ogrodje ter mu določili ime. Tega bomo s pomočjo razreda MongoClient iz knjižnice Pymongo povezali z našo podatkovno bazo MongoDB, ki deluje na lokalnem sistemu na vratih 27017. V podatkovni bazi bomo ustvarili tudi shemo, kjer se bodo shranjevali dogodki. Ker bomo na uporabniškem vmesniku uporabljali JavaScript AngularJS ogrodje, bomo s pomočje knjižnice Triangle omogočili AngularJS notacijo v Flask predlogah.

```
app = Flask(__name__)

client = MongoClient(
    'localhost',
    27017
)
db = client.ambilight

Triangle(app)
```

Programska koda 6.4: Inicializacija podatkovne baze in Flask ogrodja.

6.3.4 Globalne spremenljivke

V našem zalednem sistemu imamo definirane tri globalne spremenljivke (programska koda 6.5), ki jih bomo uporabljali skozi celoten program, in sicer:

- HEX,
- HEX2,
- time_format.

Globalna spremenljivka HEX vsebuje vsa števila ter črke, ki se uporabljajo pri šestnajstiškem zapisu.

Druga spremenljivka se imenuje HEX2 v kateri shranjujemo slovar, ki ima v ključih shranjene vse vrednosti šestnajstiškega trojčka. S pomočjo ključev lahko dostopamo do RGB vrednosti, s katerimi nastavimo ustrezno barvo na naši namizni lučki.

Tretja in tudi zadnja globalna spremenljivka z imenom time_format je namenjena shranjevanju časovnega formata, ki ga uporabljamo tekom programa.

```
HEX = '0123456789abcdef'
HEX2 = dict((a+b, HEX.index(a)*16 + HEX.index(b)) for a in HEX
    for b in HEX)
time_format = "%H:%M:%S"
```

Programska koda 6.5: Inicializacija globalnih spremenljivk.

6.3.5 Pomožna funkcija 'rgb'

V sistemu smo deklarirali tudi pomožno funkcijo `rgb` (programska koda 6.6), ki kot argument dobi šestnajstiški trojček. Njegove črke najprej pretvorimo v male. Nato s pomočjo slovarja, ki je predstavljen s spremenljivko HEX2, za vsak trojček poiščemo ustrezno RGB vrednost, ki jo kot vrednost dodelimo k ustreznemu ključu v novem slovarju, ki je hkrati tudi rezultat naše funkcije. Naša pomožna funkcija torej šestnajstiški trojček pretvori v RGB barvni model, ki ga funkcija vrne kot slovar.

```
def rgb(triplet):
    triplet = triplet.lower()
    return {
        'R': HEX2[triplet[0:2]],
        'G': HEX2[triplet[2:4]],
        'B': HEX2[triplet[4:6]]
    }
```

Programska koda 6.6: Pomožna funkcija z imenom rgb.

Sledil bo ključni del našega zalednega sistema. To je razred Led, ki s pomočjo svojih metod, ki jih bomo opisali v nadaljevanju, krmili našo namizno lučko.

6.3.6 Razred Led

V programski kodi 6.7 smo definirali razred `Led` ter uporabili funkcijo `__init__`, ki se izvede ob inicializaciji razreda.

```
class Led:
    def __init__(self):
        self.red_pin = 18
        self.green_pin = 17
        self.blue_pin = 22
        self.pins = [self.red_pin,self.green_pin,self.blue_pin]

        self.queue = Queue.Queue()
        self.mode = 'circadian'
        self.led_state = [0, 0, 0]
        self.cycle_tasks = []
        self.run = True

        self.led_loop = Thread(target=self.loop)
        self.led_loop.start()
        self.circadian_mode()
```

Programska koda 6.7: Inicializacija razreda Led.

Zaradi tega smo v tej funkciji najprej za vsako izmed barv določili številko priključka, ki ga uporablja GPIO vmesnik. Številke priključkov smo zaradi

hitrejšega in lažjega pisanja programske kode dodali še v tabelo. V nadaljevanju smo inicializirali vrsto in barve, ki se bodo uporabljale ob zagonu, ime načina uporabe in tabelo, ki bo vsebovala dogodke, katere bomo uporabljali pri načinu prelivanja barv. Poleg tega smo definirali nit ter določili funkcijo, ki bo tekla na niti. Za konec smo določili še privzeti način uporabe, in sicer način cirkadianega ritma.

```
def loop(self):
    while self.run:
        led_event = self.queue.get(block=True)
        self.set_rgb(led_event)
        self.led_state = led_event
```

Programska koda 6.8: Funkcija loop z neskončno zanko.

Funkcija z imenom `loop` (programska koda 6.8), je funkcija s pomočjo katere ob pravem času nastavimo ustrezno barvo naše namizne lučke. Funkcija je sestavljena iz neskončne zanke, kjer v vsakem obhodu iz vrste vzamemo tabelo z RGB barvnim modelom, če le-ta obstaja. Nato RGB vrednosti zapišemo v spremenljivko `led_state`, s pomočjo funkcije `set_rgb` (programska koda 6.9) pa nastavimo ustrezno barvo lučke.

```
def set_rgb(self, state):
        for i, value in enumerate(state):
            os.system(
                'sudo echo {pin}={rgb_value} > /dev/pi-blaster'.
    format(
                    pin=self.pins[i],
                    rgb_value=(float(value * 100) / 255) / 100
                )
            )
```

Programska koda 6.9: Funkcija set_rgb za nastavitev barve LED traku.

Zgoraj si lahko ogledamo funkcijo `set_rgb`, ki kot argument dobi tabelo z RGB barvnim modelom, čez katero se sprehodimo z uporabo for zanke. S pomočjo funkcije `enumerate` v vsakem obhodu dobimo vrednost posamezne barve in trenutni indeks, s katerim iz tabele `pins` za vsako izmed barv RGB

modela dobimo številko priključka, ki ga uporablja GPIO vmesnik. Ker bomo uporabili program Pi-blaster, ki sprejema samo vrednosti od 0 do 1, moramo RGB vrednost pretvoriti s pomočjo naslednjega izraza:

$$f = \frac{\frac{RGB\_vrednost*100}{255}}{100}. \tag{6.1}$$

```
sudo echo {pin}={rgb_value} > /dev/pi-blaster
```

Programska koda 6.10: Ukaz za nastavitev PWM vrednosti na izbranem priključku.

Dobljeno vrednost in številko priključka nato vnesemo v ukaz (programska koda 6.10), ki se izvede v ukazni vrstici ter izhod izbranega priključka nastavi na PWM obratovalni cikel izračunane RGB vrednosti, ki jo moramo pomnožiti s 100. S pomočjo tega tudi spreminjamo barve na naši lučki.

```
def event_mode(self):
    self.mode = 'events'
    current_time = datetime.time(
        datetime.now(
            pytz.timezone('Europe/Ljubljana')
        )
    )
    for event in db.events.find({}):
        start_time = datetime.time(
            datetime.strptime(
                event['event_start_time'],
                time_format
            )
        )
        end_time = datetime.time(
            datetime.strptime(
                event['event_end_time'],
                time_format
            )
```

```
        )
        if current_time > start_time and current_time < end_time
    :
            self.auto_transition(
                state=event['event_state'],
                mode='events'
            )
            break
```

Programska koda 6.11: Funkcija event_mode za izbiro dogodkovnega načina.

Funkcija `event_mode` (programska koda 6.11) se uporablja pri načinu preklapljanja dogodkov, katerega tudi nastavimo v prvi vrstici funkcije. Nato s pomočjo funkcij `datetime.time`, `datetime.now` in `pytz.timezone` pridobimo trenutni čas glede na časovni pas Ljubljane. Z ukazom `db.events.find()` iz podatkovne baze pridobimo obstoječe dogodke, preko katerih se sprehodimo s pomočjo for zanke. Iz vsakega dogodka izluščimo začetni in končni čas. V kolikor je trenutni čas večji od začetnega in manjši od končnega časa dogodka poženemo funkcijo `auto_transition` (programska koda 6.12), ki se nahaja v razredu `Led`. Ta kot argumente funkcije prejme RGB barvni model in trenutno ime načina uporabe. Za konec še prekinemo izvajanje for zanke.

Funkcija, ki se uporablja pri načinu cirkadianega ritma, je skoraj enaka kot funkcija `event_mode`. Razlikuje se le v tem, da nastavimo drugačen način uporabe ter da dogodke ne pridobimo iz podatkovne baze, ampak jih imamo definirane v naši Python datoteki. Pri ujemanju dogodkov prav tako sproži funkcijo `auto_transition`, ki dobi tri argumente, in sicer:

- self,
- *args,
- **kwargs.

Zadnja dva sta posebna, saj se uporabljata v primerih, ko ne vemo, koliko argumentov bo imela naša funkcija, oziroma ne vemo njihovih imen. Sledeča argumenta posredujemo funkciji `transition` (programska koda 6.12), ki iz argumenta `**kwargs` izlušči sledeče argumente:

- state,
- mode,
- transition_duration.

Argument `state` vsebuje RGB barvni model, `mode` je trenutni način uporabe, v argumentu `transition_duration` pa navedemo število barv, ki se bodo uporabile pri prelivanju med zadnjo barvo ter barvo, ki jo bomo nastavili. V funkciji najprej izpraznimo vrsto ter s pomočjo pogojnega stavka preverimo, ali je trenutni način uporabe enak tistemu, ki smo ga dobili v argumentu. Če je način enak, se skozi for zanko sprehodimo tolikokrat, kolikor barv želimo prikazati v prelivu. V for zanki nato vse barve dodamo v vrsto. Za konec v vrsto dodamo tudi barvni model, ki smo ga dobili kot argument.

Ne smemo pozabiti, da funkcija `loop` deluje na niti ter da vsebuje neskončno zanko, kjer ob vsaki spremembi vrste pridobimo barvi model, ki ga nato posredujemo funkciji `set_rgb` s pomočjo, katere nastavimo barvo na naši lučki.

```
def auto_transition(self, *args, **kwargs):
    self.transition(*args, **kwargs)

def transition(self,
               state,
               mode,
               transition_duration=20):
    with self.queue.mutex:
        self.queue.queue.clear()

    if(mode == self.mode):
        for transition_count in range(transition_duration - 1):
            event_state = []
            for component in range(3):
                event_state.append(
                    self.led_state[component] +\
                    (state[component] - self.led_state[component
    ]) *\
```

```
                    transition_count / transition_duration)
            self.queue.put(event_state)

        self.queue.put(state)
```

Programska koda 6.12: Funkciji za prelivanje barv.

6.3.7 Uporaba razreda Led in BackgroundScheduler

Razred `Led` bo v nadaljevanju uporabljala spremenljivka z imenom `led_class`. Poleg tega bomo inicializirali tudi spremenljivko z imenom `schedule`, preko katere bomo dostopali do funkcij iz razreda `BackgroundScheduler`. S pomočjo funkcij tega razreda bomo dodali dogodke, ki vsebujejo začetni ter končni čas, funkcijo ter njene morebitne argumente. Razred bo nato poskrbel za proženje dodeljenih funkcij s pripadajočimi argumenti, ki se bodo sprožile ob začetnem času dogodkov. V programski kodi 6.13 lahko vidimo inicializacijo omenjenih spremenljivk ter dodajanje dogodkov, ki jih dobimo iz podatkovne baze v razred `BackgroundScheduler` s pomočjo funkcije `add_job`.

```
led_class = Led()
schedule = BackgroundScheduler()
schedule.start()

event_tasks = {}

for event in db.events.find({}):
    start_hour = event['event_start_time'].split(':')[0]
    start_minute = event['event_start_time'].split(':')[1]
    start_second = event['event_start_time'].split(':')[2]
    start_time = datetime.strptime(event['event_start_time'],
   time_format)
    end_time = datetime.strptime(event['event_end_time'],
   time_format)
    event_duration = ((end_time - start_time).seconds if (
   end_time - start_time).seconds > 1 else 1)
    event_tasks[str(event['_id'])] = schedule.add_job(
        led_class.auto_transition,
```

```
        'cron',
        hour=start_hour,
        minute=start_minute,
        second=start_second,
        name=str(event['_id']),
        kwargs={
            'state': event['event_state'],
            'mode': 'events',
            'transition_duration': event['transition_duration']
        },
        misfire_grace_time=event_duration
    )
```

Programska koda 6.13: Dodajanje dogodkov v razred BackgroundScheduler.

Kot smo že prej omenili, dogodke pridobimo iz podatkovne baze ter se čez njih sprehodimo z uporabo for zanke. Iz vsakega dogodka izluščimo uro, minuto ter sekunde, ki predstavljajo začetni čas in jih uporabimo v funkciji `add_job`. V sledeči funkciji uporabimo tudi končni čas, ki v funkciji predstavlja `misfire_grace_time`. To je čas do katerega se še lahko izvede dogodek, v kolikor je v prejšnjih poizkusih prišlo do napake. V funkciji določimo še dodatno funkcijo, ki se bo izvedla ter njene argumente. V našem primeru je to funkcija razreda `Led`, in sicer `auto_transition`. Za katero smo z izborom načina `cron` določili, da se za izbran dogodek izvede enkrat dnevno.

6.3.8 Pogledi v ogrodju Flask

Sedaj pridejo na vrsti pogledi, ki pošiljajo podatke našemu uporabniškemu vmesniku. Programsko kodo osnovnega pogleda, ki v brskalnik vrne vsebino datoteke index.html, si lahko ogledamo v programski kodi 6.14.

```
@app.route('/')
def index():
    return render_template('index.html')
```

Programska koda 6.14: Osnovni pogled za vsebino datoteke index.html.

Kot vidimo, je vrstico pred deklaracijo funkcijo dekorator `@app.route('/')`, s pomočjo katerega definiramo, da gre za pogled. Do njega dostopamo tako, da v spletnem naslovu domeni pripnemo poševnico. Rezultat pogleda je HTML predloga z imenom `index`.

```
@app.route('/mode/events')
def event_mode():
    if led_class.mode is not 'events':
        led_class.event_mode()
    return jsonify({'mode': "events"})
```

Programska koda 6.15: Pogled za nastavitev dogodkovnega načina uporabe.

Naredili smo tudi poglede s katerimi lahko spremenimo način izvajanja. V programski kodi 6.15 se nahaja primer, ko naši lučki nastavimo način preklapljanja med dogodki. V funkciji najprej preverimo, ali naša lučka ni v načinu preklapljanja med dogodki. Če ta pogoj drži, potem prožimo funkcijo `event_mode` iz našega `Led` razreda. V uporabniški vmesnik vrnemo slovar v JSON obliki. Podoben pogled smo naredili tudi za način cirkadianega ritma. Ker lahko naše dogodke tudi dodajamo, urejamo ter brišemo, smo bili primorani narediti poglede, ki omogočajo te funkcije.

```
@app.route('/get/events')
def get_events():
    return dumps(db.events.find().sort("event_start_time", 1))

@app.route('/get/current_mode')
def get_mode():
    return jsonify({'mode': "%s" % led_class.mode})
```

Programska koda 6.16: Pogled za pridobitev dogodkov in trenutnega načina uporabe.

V programski kodi 6.16 imamo dva pogleda. Prvi pogled je funkcija z imenom get_events, ki nam vrne vse dogodke iz podatkovne baze, urejene po začetnem času. Medtem nam drugi pogled vrne trenutno aktivni način uporabe.

```
event = {
  "event_name": event.get("event_name"),
  "event_start_time": event.get("event_start_time") + ":00",
  "event_end_time": event.get("event_end_time") + ":00",
  "transition_duration": event.get("transition_duration"),
  "event_state": event.get("event_state")
}

_id = db.events.save(event)
```

Programska koda 6.17: Shranjevanje dogodka v podatkovno bazo.

Pogled z imenom `add_edit_event` je najobsežnejši pogled v naši programski kodi. Uporabljamo ga za dodajanje in urejanje dogodkov. Pogled deluje tako, da iz uporabniškega vmesnika pošljemo dogodek, ki ga želimo urediti ali dodati. Nato v programski kodi opravimo validacijo v kateri preverimo, če se začetni ali končni čas prekriva z ostalimi dogodki, ali so vsa polja v pravem formatu ter če so zahtevana polja izpolnjena. Če je validacija uspela preverimo, ali dogodek vsebuje enolični identifikator. Če ga, dogodek uredimo. V nasprotnem primeru ga dodamo v podatkovno bazo (programska koda 6.17). Če smo dodali nov dogodek, moramo le-tega dodati v razred `BackgroundScheduler`, v nasprotnem primeru pa samo uredimo njegov začetni čas.

```
@app.route('/delete/event', methods=['POST'])
def delete_event():
  event = request.json.get('event', {})
  if event.get("_id"):
      db.events.remove({"_id": ObjectId(event.get("_id").get("
    $oid"))})
      event_tasks.get(event.get("_id").get("$oid")).remove()
      del event_tasks[event.get("_id").get("$oid")]

      return jsonify({'status': "success"})
  return jsonify({'status': "failed"})
```

Programska koda 6.18: Brisanje dogodka iz podatkovne baze.

Dogodke lahko brišemo s pomočjo pogleda `delete_event` (programska koda 6.18). Isto kot pri dodajanju ali urejanju dogodka, tudi pri brisanju iz uporabniškega vmesnika dobimo dogodek. Nato preverimo, če le-ta vsebuje enolični identifikator. V primeru, da ga ne, vrnemo slovar v JSON obliki, ki vsebuje sporočilo, da brisanje ni uspelo. V nasprotnem primeru dogodek izbrišemo iz podatkovne baze in razreda `BackgroundScheduler` ter vrnemo slovar v JSON obliki s potrditvenim sporočilom.

```
led_class.cycle_tasks.append(
  schedule.add_job(
      led_class.auto_transition,
      'interval',
      seconds=40,
      start_date=datetime.now() + timedelta(seconds=1),
      name='__cycle_0',
      kwargs={'state': [126, 0, 255], 'mode': 'cycle', '
   transition_duration': 800}
  )
)
```

Programska koda 6.19: Izsek programske kode iz pogleda cycle_mode.

Način prelivanja barv nastavimo v pogledu z imenom cycle_mode. Pogled deluje tako, da v razred `BackgroundScheduler` dodajmo dogodke, ki se izvajajo v intervalu. Za vsak interval moramo določiti začetni čas in trajanje intervala. Primer dodajanje dodajanja dogodka v razred `BackgroundScheduler`, kjer prožimo funkcijo `auto_transition` si lahko ogledamo v programski kodi 6.19.

```
def manual_set(hex_val):
    return_object = {
        'output': None,
        'error': None,
        'success': False
    }
    if hex_val == '000000':
        led_class.mode = 'stop'
        led_class.queue.put([0, 0, 0])
```

```
    else:
        led_class.mode = 'manual'
        rgb_val = rgb(hex_val)
        led_class.transition(
            [
                rgb_val['R'],
                rgb_val['G'],
                rgb_val['B']
            ],
            'manual'
        )

    return_object['success'] = True
    return return_object

@app.route('/set/<hex_val>')
def set_hex_val(hex_val):
    return jsonify(manual_set(hex_val))

@app.route('/mode/off')
def off_mode():
    manual_set('000000')
    return jsonify({'mode': "stop"})
```

Programska koda 6.20: Pogled za nastavitev barve in izklop luči.

Ostala sta nam še zadnja dva pogleda, in sicer `set_hex_val` in `off_mode`. Oba pogleda, ki sta opisana v programski kodi 6.20, uporabljata funkcijo `manual_set`, ki prejme argument v šestnajstiškem zapisu. V primeru, da je šestnajstiški zapis enak 000000, luč ugasne. V nasprotnem primeru funkciji `rgb` posredujemo šestnajstiški zapis, kot rezultat pa dobimo RGB barvni model. Tega poleg imena način uporabe posredujemo funkciji `transition`. Pogled `set_hex_val` torej uporabljamo za ročno nastavljanje barve na naši lučki medtem, ko pogled `off_mode` našo lučko ugasne.

6.4 Uporabniški vmesnik

V tem podpoglavju bomo opisali arhitekturo in razvoj uporabniškega vmesnika ter implementacijo mobilne aplikacije.

6.4.1 Arhitektura uporabniškega vmesnika

Kot vemo, nam osnovni pogled zalednega sistema na spletnem naslovu domene prikaže vsebino datoteke `index.html`, ki je hkrati tudi ključna datoteka uporabniškega vmesnika. Ker HTML datoteke omogočajo le prikazovanje vsebine, moramo v glavo datoteke vključiti tudi stilske predloge, ki se nahajajo v datotekah s končnicami CSS. Za funkcionalnosti pa skrbijo JavaScript datoteke, ki uporabljajo knjižnico jQuery ter ogrodji Framework7 in AngularJS.

6.4.2 Razvoj uporabniškega vmesnika

Pri razvoju uporabniškega vmesnika smo najprej kreirali HTML datoteko z imenom `index.html`, v katero smo vključili stilske predloge ter JavaScript datoteke.

```
<link href="/static/css/framework7.css" rel="stylesheet">
<link href="/static/css/style.css" rel="stylesheet">
<link href="/static/css/animations.css" rel="stylesheet" type="
    text/css">
<link href="/static/css/jquery.minicolors.css" rel="stylesheet">
```

Programska koda 6.21: Uvoz stilskih predlog v html datoteko.

Stilske predloge iz programske kode 6.21 smo vključili v glavo HTML datoteke medtem, ko smo JavaScript datoteke (programska koda 6.22) vključili tik pred koncem `body` značke. Med vključenimi datotekami se nahajajo tudi knjižnica jQuery in ogrodji AngularJS ter Framework7.

```
<script src="/static/js/jquery-1.10.1.min.js" type="text/
    javascript"></script>
```

```
<script src="/static/js/jquery.validate.min.js" type="text/
    javascript"></script>
<script src="/static/js/framework7.js" type="text/javascript"></
    script>
<script src="/static/js/my-app.js" type="text/javascript"></
    script>
<script src="/static/js/jquery.minicolors.min.js" type="text/
    javascript" charset="utf-8"></script>
<script src="/static/js/app/angular.min.js" type="text/
    javascript"></script>
<script src="/static/js/app/angular-animate.min.js" type="text/
    javascript"></script>
<script src="/static/js/app/angular-resource.min.js" type="text/
    javascript"></script>
<script src="/static/js/app/angular-app.js" type="text/
    javascript"></script>
<script src="/static/js/app/angular-directives.js" type="text/
    javascript"></script>
<script src="/static/js/app/angular-services.js" type="text/
    javascript"></script>
<script src="/static/js/app/angular-controllers.js" type="text/
    javascript"></script>
```

Programska koda 6.22: Uvoz JavaScript datotek v html datoteko.

Ogrodje Framework7 smo uporabili zaradi boljše uporabniške izkušnje, saj vključuje animacije ter gradnike mobilnega operacijskega sistema Android. Animacije se izvajajo med preklapljanjem pogledov. Te definiramo v HTML datoteki tako, da v HTML znački dodamo atribut `data-page="index"`. Tako smo definirali pogled `index`, do katerega lahko dostopamo prek povezave `#index`. Ne pozabimo, da funkcije začnejo delovati šele z inicializacijo ogrodja, kar storimo v datoteki `my-app.js` s pomočjo programske kode 6.23.

```
var myApp = new Framework7({
  animateNavBackIcon: true,
  precompileTemplates: true,
  swipeBackPage: true,
  swipeBackPageThreshold: 1,
```

```
  swipePanel: "left",
  swipePanelCloseOpposite: true,
  pushState: true,
  pushStateRoot: undefined,
  pushStateNoAnimation: false,
  pushStateSeparator: '#!/',
    template7Pages: true
});

var $$ = Dom7;
var mainView = myApp.addView('.view-main', {
    dynamicNavbar: false,
    domCache: true
});
```

Programska koda 6.23: Inicializacija ogrodja Framework7.

Potem ko smo definirali vse poglede, smo s pomočjo ogrodja AngularJS implementirali funkcionalnosti in vzpostavili povezavo z zalednim sistemom, katero lahko vzpostavimo z dvema vgrajenima funkcijama iz ogrodja AngularJS, in sicer `$http` in `$resource`. V našem primeru smo se odločili, da bomo za povezave na poglede zalednega sistema, ki vračajo vsebino, uporabili funkcijo `$resource`. Za poglede, ki vračajo samo statusno kodo http odgovora pa `$http`. Razlika med njima je, da lahko s pomočjo funkcije `$resource` lažje naredimo `POST` in `GET` zahteve na zaledni sistem. Povezave, ki uporabljajo funkcijo `$resource` smo implementirali znotraj storitve `AppService`, ki smo jo naredili s pomočjo vgrajene AngularJS funkcije `factory`. Ta se nahaja v datoteki `angular-services.js`, katere kodo si lahko ogledamo0 v programski kodi 6.24.

```
var app = angular.module('app', ['ngResource']);

app.factory('AppService', function($resource){
    return {
        getMode: function () {
            return $resource('/get/current_mode');
        },
```

```
        getEvents: function () {
            return $resource('/get/events');
        },
        deleteEvent: function () {
            return $resource('/delete/event');
        },
        event: function () {
            return $resource('/add_edit/event');
        }
    }
});
```

Programska koda 6.24: Inicializacija ogrodja AngularJS in storitev AppService.

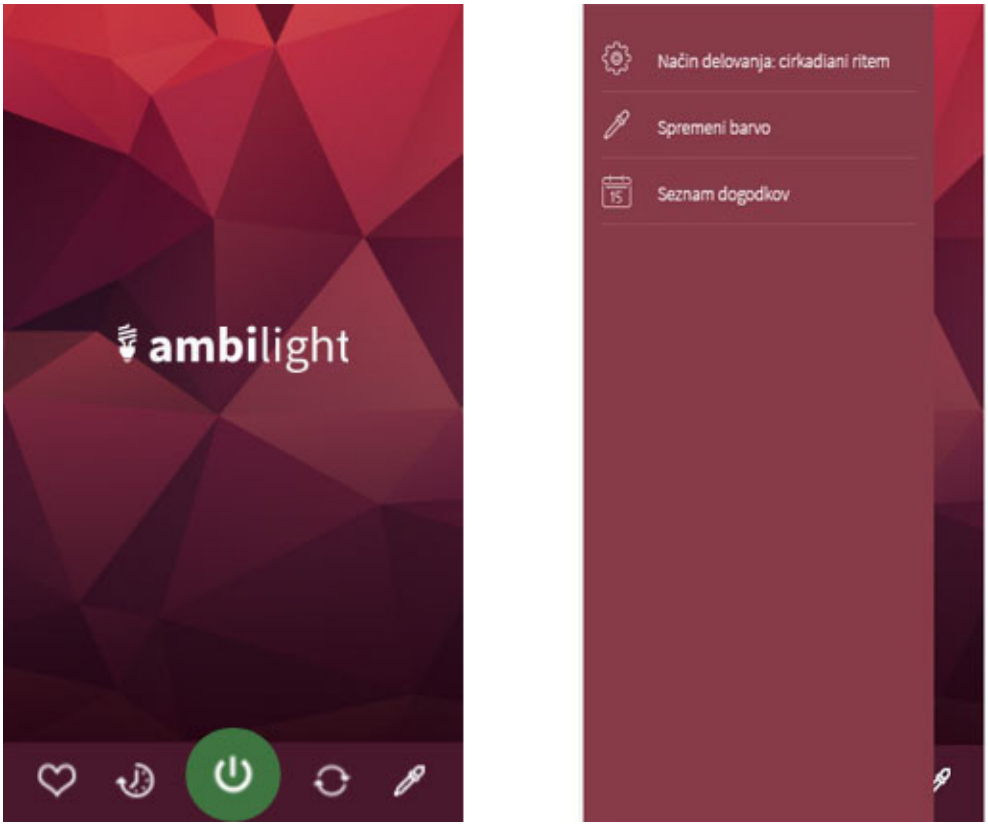
Jedro naših funkcionalnosti se nahaja v krmilniku ogrodja AngularJS, ki smo ga imenovali `AppController`. Ta je definiran s pomočjo JavaScript konstruktorja in nam omogoča manipuliranje DOM-a, kamor krmilnik vključimo s pomočjo vgrajene AngularJS direktive `ng-controller`. S tem se bo v krmilniku ustvaril nov objekt z imenom `scope` v katerem shranjujemo spremenljivke. Le-tega moramo v objekt vključiti z imenom `$scope`. Poleg tega smo v krmilnik vključili tudi storitev `AppService` ter funkciji `$http` in `$window`, ki ju uporabljamo za dostop do pogledov zalednega sistema ter preusmerjanje uporabnika. Storitev `AppService` uporabimo že na začetku našega krmilnika, in sicer v funkciji `update_state` s pomočjo, katere dobimo ime trenutnega načina delovanja. V programski kodi 6.25 si lahko ogledamo primer klica storitve `AppService`.

```
$scope.update_state = function() {
  AppService.getMode().get(function (response) {
      $scope.mode = response.mode;
  });
};
```

Programska koda 6.25: Funkcija update_state za pridobitev načina uporabe.

Pri klicu storitve `AppService` uporabimo metodo `get` s pomočjo, katere pridobimo rezultat. To je v našem primeru način uporabe, ki ga nato shra-

nimo v spremenljivko z imenom `mode`, ki jo izpišemo v stranskem meniju našega uporabniškega vmesnika. Le-tega odpremo tako, da na mobilni napravi s prstom podrsamo na desno stran, medtem ko na računalniku kliknemo na levi gumb ter miško premaknemo desno. Izpis spremenljivke v desnem stranskem meniju si lahko ogledamo na sliki 6.3.



Slika 6.3: Uvodna stran uporabniškega vmesnika ter stranski meni.

Luč lahko prižgemo ali ugasnemo s klikom na srednji gumb v orodni vrstici uvodnega pogleda uporabniškega vmesnika, kjer s pomočjo vgrajene AngularJS direktive `ng-click` izvedemo funkcijo `powerButton` (programska koda 6.26). Ta funkcija najprej preveri, ali je luč v mirovanju. Če je, s pomočjo funkcije `$http.get` dostopamo do relativnega spletnega naslova `/mode/circadian` in s tem aktiviramo način cirkadianega ritma. V primeru, da luč ni v načinu mirovanja prav tako uporabimo funkcijo `$http.get`, ven-

dar tokrat dostopamo do relativnega naslova `/mode/off` in s tem ugasnemo luč. Za konec pokličemo še funkcijo `update_state`.

```
$scope.powerButton = function(e) {
  if($scope.mode == "stop")
    $http.get("/mode/circadian");
  else
    $http.get("/mode/off");
  $scope.update_state();
};
```

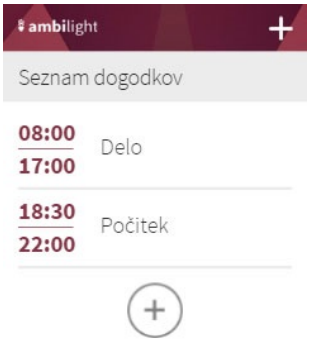
Programska koda 6.26: Funkcija za vklop ali izklop luči.

Slika 6.4: Orodna vrstica uporabniškega vmesnika.

Preostali gumbi v orodni vrstici (slika 6.4) so namenjeni spreminjanju načina delovanja. S klikom na prvo ikono z leve, luč nastavimo v način cirkadianega ritma. Druga ikona z leve aktivira dogodkovni način uporabe medtem, ko s pomočjo drugega gumba z desne, aktiviramo prelivanje med barvami. S kliki na omenjene gumbe se aktivirajo ustrezne funkcije, ki imajo le eno razliko in sicer relativni spletni naslov, ki ga posredujemo funkciji `$http.get`. Prav tako v vsaki izmed funkcij pokličemo funkcijo `update_state`. Drugačna funkcija se uporablja le za spreminjanje barve. S klikom na skrajno desni gumb v orodni vrstici uvodnega uporabniškega vmesnika najprej odpremo nov pogled, kjer se nahaja vmesnik za izbiro barve. Vmesnik, ki ga prikažemo s pomočjo `jQuery` knjižnice `minicolorsq`, nam omogoča izbiro poljubne barve. Ko le-to izberemo se izvede funkcija, ki našemu zalednemu sistemu posreduje barvo v šestnajstiškem zapisu.

Naš uporabniški vmesnik vsebuje pogled, na katerem so izpisani dogodki (slika 6.5), ki se uporabljajo pri dogodkovnem načinu delovanja. Dogodke v obliki tabele, katero pridobimo iz zalednega sistema z uporabo storitve

`AppService` in jo shranimo v spremenljivko `events`. Podrobnosti vsakega dogodka dobimo z uporabo vgrajene AngularJS direktive `ng-repeat`, saj nam le-ta omogoča sprehod čez tabelo podatkov.



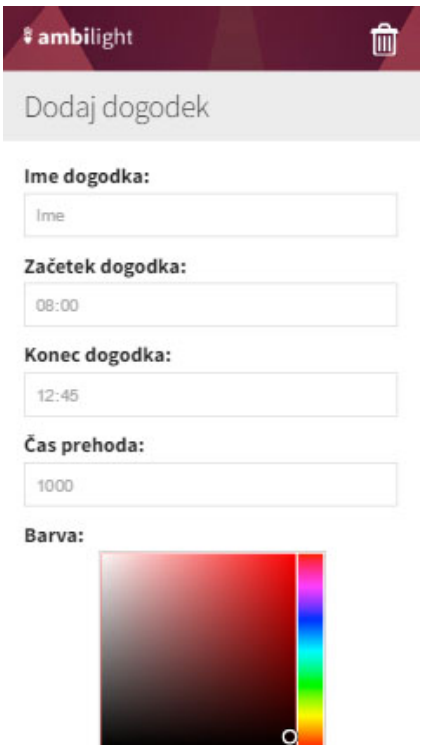
Slika 6.5: Seznam dogodkov.

Pogled za dodajanje dogodka (slika 6.6) se nam odpre, ko na pogledu, kjer so izpisani dogodki, kliknemo na plus ikono. V primeru, da na dogodek kliknemo, se nam odpre pogled za urejanje dogodka. Vsak dodan ali urejen dogodek mora vsebovati ime, začetek in konec dogodka ter barvo, medtem ko je privzeti čas prehoda dogodka traja eno sekundo, le-tega lahko tudi ponastavimo, če izpolnimo polje čas prehoda. Dogodek shranimo s klikom na gumb, ki nas prestavi na seznam dogodkov. S klikom na gumb se izvrši tudi programska koda 6.27.

```
$scope.updateOrAddEvent = function (event) {
    if(event.transition_duration == undefined)
        event.transition_duration = 1000;
    AppService.event().save({event: event}, function (response)
   {
        if(response.status == "success") {
            $scope.errors = {};
            $scope.event = {
                event_state: []
            };
            $scope.loadEvents();
            $window.location.href = '/#!/#events';
```

```
        } else {
            $scope.errors = response.errors;
        }
    }, function (error) {
        console.log('error', error);
    });
};
```

Programska koda 6.27: Dodajanje in urejanje dogodka.



Slika 6.6: Urejanje dogodka.

6.4.3 Mobilna aplikacija

Zaradi lažjega upravljanja luči smo se odločili, da bomo naš vmesnik s pomočjo spletnega pogleda vključili v mobilno aplikacijo, ki bo delovala na operacijskem sistemu Android. Pri izdelovanju aplikacije smo uporabljali

programsko orodje Android Studio, ki je namenjen prav razvijanju aplikacij za operacijski sistem Android. Pri razvoju smo najprej ustvarili datoteko `activity_main.xml`, kjer smo določili, da bo naš spletni pogled na voljo čez celoten ekran. V datoteki `MainActivity.java` smo nato določili spletni naslov na katerem se nahaja naš uporabniški vmesnik, poleg tega smo omogočili tudi izvajanje JavaScripta na naši aplikaciji. Konfiguracijo spletnega pogleda si lahko ogledamo v programski kodi 6.28.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    String url = "http://192.168.1.199:5000";

    WebView webView = (WebView) this.findViewById(R.id.webView);
    webView.getSettings().setJavaScriptEnabled(true);
    webView.loadUrl(url);
}
```

Programska koda 6.28: Inicializacija spletnega pogleda v mobilni aplikaciji.

6.5 Strojna oprema

Pri razvoju strojnega dela smo uporabili shemo (slika 6.7), ki smo jo pridobili iz interneta in vsebuje LED RGB trak, ki ga napaja 12V napajalnik, razvojno ploščico, računalnik Raspberry PI 2 in tri tranzistorje MOSFET. LED RGB trak sestavlja 60 segmentov na meter. Vsak izmed njih vsebuje 3 LED diode in porabi 60mA električnega toka. To pomeni, da vsak meter na RGB LED traku porabi 1,2A. Ker imamo pet metrov dolg RGB LED trak, na katerem je 300 LED diod, vsak izmed tranzistorjev pa mora prenesti 1A električnega toka, smo se odločili, da bomo uporabili tranzistorje MOSFET z napetostjo ponora 60V in tokom ponora 16A. Na vsakem izmed tranzistorjev smo ponor povezali na negativni potencial razvojne ploščice. Vrata

posameznega tranzistorja smo povezali z žico, ki predstavlja določeno barvo in prihaja iz LED traka. Na tranzistorju nam je ostala še ena nožica, in sicer izvor, ki smo ga povezali z GPIO vmesnikom na računalniku Raspberry Pi 2. Na negativni potencial (GND) razvojne ploščice smo povezali še preostalo žico z LED RGB traku, poleg tega smo morali nanj povezati tudi Raspberry Pi računalnik [30].

Računalnik smo morali povezati še z brezžičnim omrežjem. Na le-to se povezujemo z uporabo brezžične mrežne kartice. Vendar smo morali pred tem v operacijskem sistemu Raspbian v datoteki `wpa_supplicant.conf`, ki je shranjena v mapi `/etc/wpa_supplicant` (programska koda 6.29), vnesti še ime in geslo brezžičnega omrežja. Nato smo v datoteki `interfaces` (programska koda 6.30), ki se nahaja v mapi `/etc/network`, določiti še statični IP naslov na računalniku Raspberry Pi. S tem smo namizno lučko povezali z brezžičnim omrežjem in omogočili upravljanje le-te preko spletne in mobilne aplikacije.

Težava našega sistema je, da je operacijski sistem shranjen na SD kartici. V primeru njene okvare, bo prenehal delovati tudi naš sistem. Če pride do okvare SD kartice, moramo kupiti novo ter na njo naložiti operacijski sistem, vzpostaviti brezžično dostopno točko in pognati zagonsko skripto.

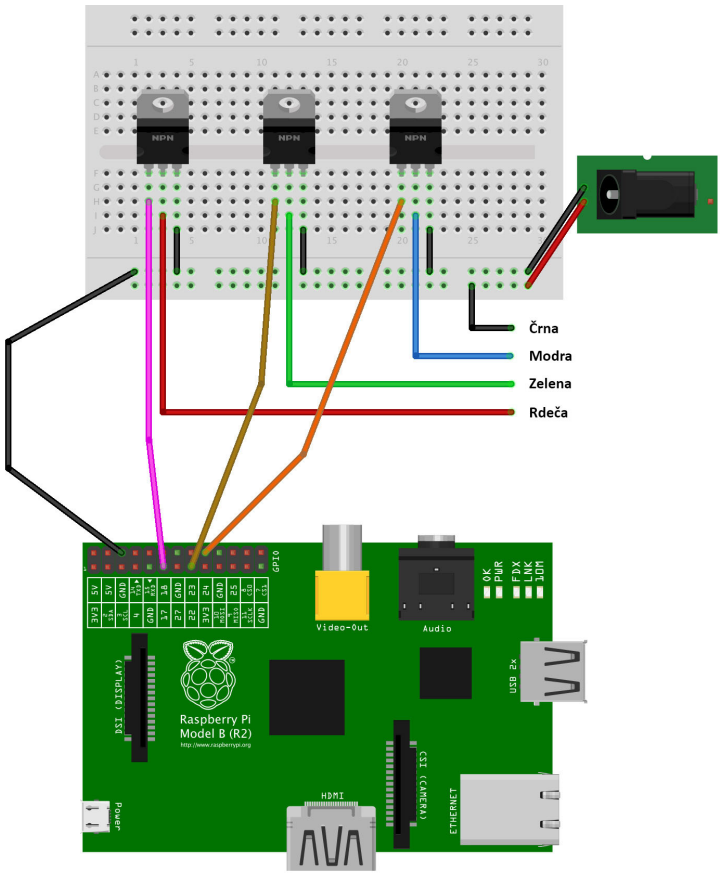
```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
network={
    ssid="pucnik"
    psk="pivola123"
    id_str="pucnik"
}
```

Programska koda 6.29: Vsebina datoteke wpa_supplicant.conf.

```
auto lo
iface lo inet loopback
iface pucnik inet static
address 192.168.1.199
netmask 255.255.255.0
network 192.168.1.0
```

```
broadcast 192.168.1.255
gateway 192.168.1.1
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

Programska koda 6.30: Vsebina datoteke interfaces.



Slika 6.7: Shema elektronskega vezja [30].

Poglavje 7

Sklepne ugotovitve

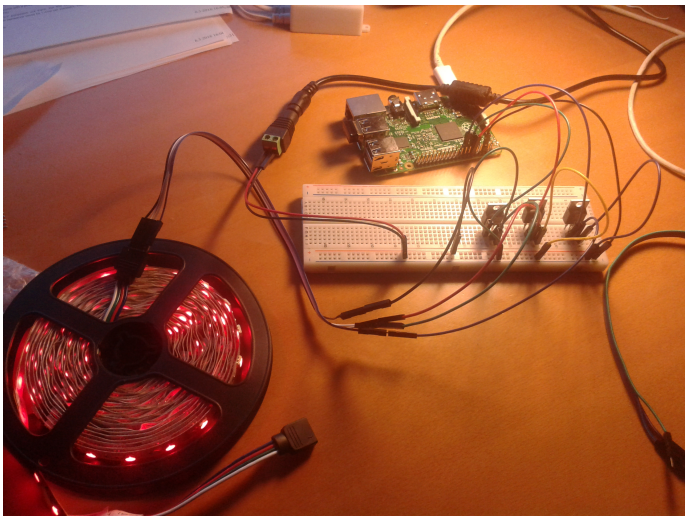
V našem diplomskem delu smo izdelali namizno lučko, ki se v realnem času odziva na uporabniške akcije. Uporabnik lahko do uporabniškega vmesnika dostopa preko spletne in mobilne aplikacije, ki deluje na operacijskem sistemu Android. Mobilna naprava pa mora biti povezana na lokalno omrežje.

Z uporabo ogrodji AngularJS in Framework7 smo izdelali uporabniški vmesnik, ki lahko v realnem času komunicira z zalednim sistemom, ki je narejen v Flask ogrodju v programskem jeziku Python. Zaledni sistem, ki deluje na računalniku Raspberry Pi, skrbi za komunikacijo s programom Pi-blaster ter deluje kot spletni strežnik.

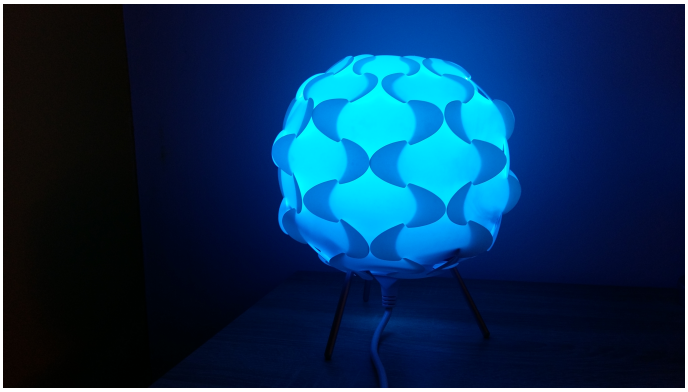
Med razvojem smo naleteli tudi na težavo. Le-ta se je pojavila pri zalednem sistemu. Želeli smo, da bi zaledni sistem deloval kot program v ozadju. V primeru, da je zaledni sistem deloval kot program v ozadju, se je prekinila povezava s programom Pi-blaster. Posledica tega je bila, da naša namizna lučka ni več prikazovala barv. Težavo smo rešili z zagonsko skripto, ki skrbi, da se zaledni sistem začne izvajati ob zagonu računalnika Raspberry Pi.

Projekt je še v zgodnji fazi razvoja. V nalogi smo izdelali delujoč prototip sistema. V prihodnosti bo potrebno spremeniti obliko lučke, ključne elemente pa bolj zaščititi. Ker se naš prototip povezuje na brezžično omrežje, moramo pred tem na računalniku Raspberry Pi vpisati ime in geslo brezžičnega omrežja. Težava nastane, saj si morajo uporabniki sami nastaviti podatke

brezžičnega omrežja v računalniku Raspberry Pi. Zaradi tega bo v prihodnosti potrebno to popraviti in omogočiti, da bodo uporabniki podatke brezžičnega omrežja vnesli v aplikacijo. Končni izdelek diplomskega dela oziroma delujoč prototip namizne lučke 'Ambilight' je prikazan na slikah 7.1 in 7.2.



Slika 7.1: Strojna oprema sistema.



Slika 7.2: Prototip namizne luči 'Ambilight'.

Literatura

[1] Španinger K, Košir R, Fink M, Debeljak N, Rozman D. Cirkadiani ritem pri ljudeh. Zdrav Vestn, 2009; 78: 651 – 657.

[2] Exposure to Room Light before Bedtime Suppresses Melatonin Onset and Shortens Melatonin Duration in Humans. [Online]. Dosegljivo: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3047226/. [Dostopano 5. 2. 2017].

[3] Evening use of light-emitting eReaders negatively affects sleep, circadian timing, and next-morning alertness . [Online]. Dosegljivo: http://www.pnas.org/content/112/4/1232.full.pdf. [Dostopano 5. 2. 2017].

[4] HTML5. [Online]. Dosegljivo: https://www.w3.org/blog/news/archives/4167. [Dostopano 5. 2. 2017].

[5] CSS developer guide. [Online]. Dosegljivo: https://developer.mozilla.org/en-US/docs/Web/Guide/CSS. [Dostopano 5. 2. 2017].

[6] ECMAScript 6 — New Features: Overview & Comparison. [Online]. Dosegljivo: http://es6-features.org. [Dostopano 5. 2. 2017].

[7] JavaScript. [Online]. Dosegljivo: https://sl.wikipedia.org/wiki/JavaScript. [Dostopano 5. 2. 2017].

[8] jQuery. [Online]. Dosegljivo: https://jquery.com. [Dostopano 5. 2. 2017].

[9] Single-page application. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Single-page_application. [Dostopano 5. 2. 2017].

[10] The DCI Architecture: A New Vision of Object-Oriented Programming. [Online]. Dosegljivo: http://www.artima.com/articles/dci_vision.html. [Dostopano 5. 2. 2017].

[11] Framework7. [Online]. Dosegljivo: https://framework7.io. [Dostopano 5. 2. 2017].

[12] Python za programerje. [Online]. Dosegljivo: http://trac.lecad.si/vaje/raw-attachment/wiki/python/pythonzaprogramerje.pdf. [Dostopano 5. 2. 2017].

[13] PEP 20 – The Zen of Python. [Online]. Dosegljivo: https://www.python.org/dev/peps/pep-0020/. [Dostopano 5. 2. 2017].

[14] Pip (package manager). [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Pip_(package_manager). [Dostopano 5. 2. 2017].

[15] Flask (web framework). [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Flask_(web_framework). [Dostopano 5. 2. 2017].

[16] Vagrant (software). [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Vagrant_(software). [Dostopano 5. 2. 2017].

[17] Android (operacijski sistem). [Online]. Dosegljivo: https://sl.wikipedia.org/wiki/Android_(operacijski_sistem). [Dostopano 5. 2. 2017].

[18] MongoDB. [Online]. Dosegljivo: https://docs.mongodb.com. [Dostopano 5. 2. 2017].

[19] Pulse-width modulation. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Pulse-width_modulation. [Dostopano 5. 2. 2017].

[20] APScheduler 3.2.0. [Online]. Dosegljivo: https://pypi.python.org/pypi/APScheduler. [Dostopano 5. 2. 2017].

[21] Sublime Text. [Online]. Dosegljivo: https://www.sublimetext.com. [Dostopano 5. 2. 2017].

[22] Android Studio. [Online]. Dosegljivo: https://developer.android.com/studio/index.html. [Dostopano 5. 2. 2017].

[23] Raspberry Pi. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Raspberry_Pi. [Dostopano 5. 2. 2017].

[24] IEEE 802.11. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/IEEE_802.11. [Dostopano 5. 2. 2017].

[25] EW-7811Un. [Online]. Dosegljivo: http://www.edimax.com/edimax/merchandise/merchandise_detail/data/edimax/global/7811un. [Dostopano 5. 2. 2017].

[26] Tranzistor. [Online]. Dosegljivo: https://sl.wikipedia.org/wiki/Tranzistor. [Dostopano 5. 2. 2017].

[27] TRANZISTOR. [Online]. Dosegljivo: http://www.s-sers.mb.edus.si/gradiva/rac/moduli/mikroprocesorji/03_sloji/03_datoteka [Dostopano 5. 2. 2017].

[28] Virtualenv. [Online]. Dosegljivo: https://virtualenv.pypa.io/en/stable/. [Dostopano 5. 2. 2017].

[29] Spletna barva. [Online]. Dosegljivo: https://sl.wikipedia.org/wiki/Spletna_barva. [Dostopano 5. 2. 2017].

[30] Controllable Christmas Lights using Raspberry Pi & RGB LED Strip. [Online]. Dosegljivo: http://aruljohn.com/blog/raspberrypichristmaslightsrgbled. [Dostopano 5. 2. 2017].

[31] Circadian rhythm. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Circadian_rhythm. [Dostopano 5. 2. 2017].

[32] Model MVC. Basic MVC example in ASP .NET MVC 4. [Online]. Dosegljivo: https://jlborntocoding.wordpress.com/2014/03/06/model-mvc-basic-mvc-example-in-asp-net-mvc-4. [Dostopano 5. 2. 2017].

[33] Pulse Width Modulation (PWM) Using NI-DAQmx and LabVIEW. [Online]. Dosegljivo: http://www.ni.com. [Dostopano 5. 2. 2017].

[34] RASPBERRY PI 2. [Online]. Dosegljivo: https://www.raspberrypi.org/blog/raspberry-pi-2-on-sale. [Dostopano 5. 2. 2017].

[35] How to configure Edimax EW-7811UN Wifi dongle on Raspbian. [Online]. Dosegljivo: https://www.andreagrandi.it/2014/09/02/how-to-configure-edimax-ew-7811un-wifi-dongle-on-raspbian. [Dostopano 5. 2. 2017].

[36] Tranzistor. [Online]. Dosegljivo: http://colos.fri.uni-lj.si/eri/racunalnistvo/rac_sloji/tranzistor.html. [Dostopano 5. 2. 2017].

[37] Vpliv svetlobe na ljudi. [Online]. Dosegljivo: http://lrf.fe.uni-lj.si/otv_do_razsvetljava/dor%20vpliv%20svetlobe%20na%20ljudi.pdf. [Dostopano 5. 2. 2017].

[38] Photic memory for executive brain responses. [Online]. Dosegljivo: http://www.pnas.org/content/111/16/6087.abstract. [Dostopano 5. 2. 2017].

[39] GNU General Public Licence. [Online]. Dosegljivo: https://www.gnu.org/copyleft/gpl.html. [Dostopano 5. 2. 2017].